Freesound Audio Tagging 2019

TAA - proyecto 2
Grupo C

Magdalena Castrillo
Elias Courdin
Uriel Yaffé

13 de julio de 2025

Índice

1. Objetivos

Familiarizarnos con los datos del desafío de Freesound Audio Tagging 2019, la métrica del problema y aplicar lo viso en el curso de redes convolucionales profundas para implementar un clasificador de etiquetas para los audios para finalmente subirlo al kaggle del curso.

2. Introducción

Este desafío se presenta como un problema de audio, ya que es el formato de los datos de entrada y el objetivo es etiquetarlos de forma correcta dentro de 80 clases posibles. Sin embargo, será abordado como un problema de imágenes estudiando los espectrogramas, es decir su representación en tiempo-frecuencia.

Un espectrograma log-Mel es una representación de cómo varía la energía de una señal de audio a lo largo del tiempo y en distintas bandas de frecuencia. Además, la energía se expresa en escala logarítmica, lo que permite que se adapte a la escala del oído humano y resaltar mejor las diferencias de energía entre frecuencias altas y bajas. A partir del espectrograma log-Mel, se pueden extraer características más compactas y relevantes conocidas como MFCCs que capturan la forma general y atributos más importantes del espectrograma.

Como se transformó el audio en una representación visual que muestra su contenido en el dominio de la frecuencia a lo largo del tiempo, nos parece razonable utilizar redes neuronales convolucionales (CNNs). Las CNNs son muy eficaces para detectar patrones espaciales y extraer características relevantes en imágenes, lo que se adapta muy bien a este tipo de representación. En los espectrogramas, estos patrones o características pueden representar transiciones de frecuencia, timbres, ritmos o estructuras sonoras que son claves para identificar distintos tipos de sonidos Además, existe una gran cantidad de arquitecturas preentrenadas de CNNs que pueden aprovecharse mediante técnicas de transferencia de aprendizaje, lo cual permite obtener buenos resultados sin necesidad de entrenar modelos desde cero y sin contar con tantos datos.

3. Análisis de los datos y preprocesamiento

Los audios disponibles son de dos tipos, curados y ruidosos. Los primeros consisten en audios bien etiquetados, los segundos contienen etiquetas menos confiables que fueran asignadas de forma automática. La cantidad de datos ruidosos es de 19.815, mucho mayor a la cantidad de datos curados, 4.970.

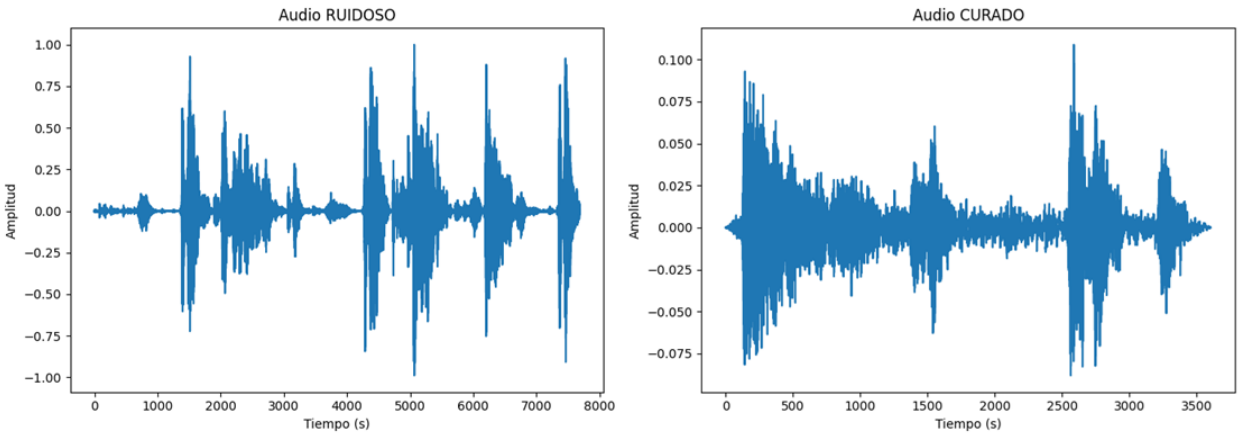

Figura 3.1: Visualización temporal de audios. A la izquierda el ruidoso, '00097e21.wav' y a la derecha el curado '0006ae4e.wav'

Como comparación vamos a tomar el primer audio de cada conjunto para analizar sus diferencias. Como se puede ver en la Figura 3.1, el audio ruidoso se encuentra más particionado que el curado que parece contener solo el audio deseado. Pudimos escucharlos a ambos y comparar con sus etiquetas. El audio curado es un perro ladrando y se distingue claramente, parece haber un regador de pasto de fondo, su etiqueta es 'Bark'. Por otro lado el ruidoso pareciera ser una mujer bañando a una bebé ya que se oye su voz "sit down" seguido de risas de bebé y sonidos de agua de fondo, en este caso la etiqueta es 'Bathtub_(filling_or_washing)'. Como se pudo observar con esta sencilla comparación los audios curados tienen sonidos más concretos y mejor etiquetados que los ruidosos.

Cada audio puede contar con más de una etiqueta, es decir, una o varias palabras que describen su contenido dentro de 80 clases posibles. En la Figura 3.2 se puede ver la distribución de las *labels* de cada uno de los conjuntos. Se destaca que el conjunto de los ruidosos contiene etiquetas muy balanceadas con unos 300 audios por cada una. En cambio en el conjunto de los curados hay cierto desbalance, siendo la etiqueta con mayor ocurrencia 'Bark' con

75 ocurrencias y la de menor es 'Accordion' con 47. Este desbalance será tenido en cuenta a la hora de conformar el conjunto de validación, ver sección 5.

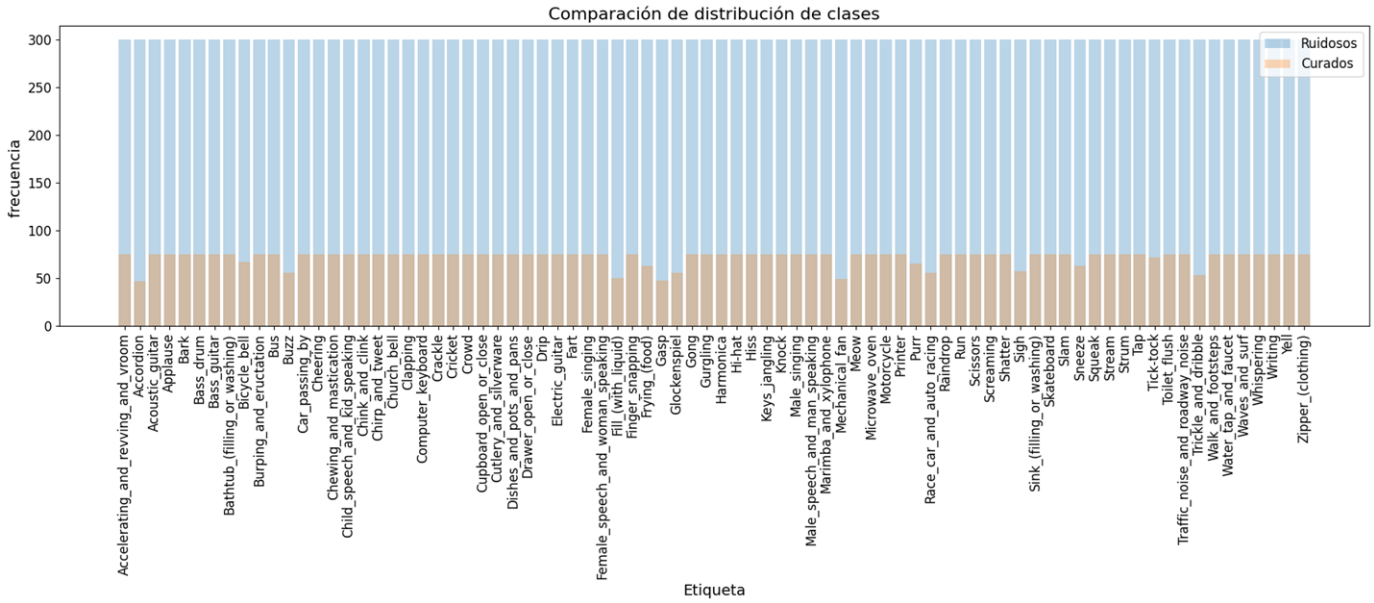

Figura 3.2: Histograma de etiquetas para el conjunto de audios ruidoso y curado del conjunto de *train*.

Como se dijo el problema es multietiqueta, si bien tanto en el conjunto de los curados como en los ruidosos, la mayoría contiene 1 etiqueta, existen casos con 2, 3 y hasta 7 etiquetas por archivo. Por este motivo agregamos una columna a los *dataframes* conteniendo un vector de largo 80 que de forma *one hot encoder* alfabético, en el cual se indica con un 1 si contiene cierta etiqueta y con un 0 si no. Esto fue implementado mediante la función `codificar_etiquetas`, adaptada de [8], aplicada a cada uno de los *dataframes*. De esta forma cada conjunto contiene la columna *labels* original y una nueva *label_vec* conteniendo el vector en formato *one hot encoder*. Este formato condice con lo esperado al realizar la *submission* como se puede corroborar en el archivo sample_submission.csv.

Como se pudo ver con el ejemplo de la Figura 3.1, la duración de los audios no es uniforme. En el caso de los curados varía entre 0,3 y 57,57 segundos, mientras que en el caso de los ruidosos entre 1 y 16 segundos. Esta duración variable genera imágenes de distintos tamaños al transformarlos en espectrogramas, lo cual representa un problema para el entrenamiento de redes convolucionales que requieren entradas de tamaño fijo. Por este motivo se realizaron dos funciones de preprocesamiento `sacar_silencios` y `largo_fijo`.

La primera utiliza la función `split_on_silence` de la librería `pydub`, adaptada de [6]. En el caso de los audios curados se usaron parámetros dependiendo del largo del audio y en el caso de los ruidosos parámetros fijos. Como esta función usa librerías incompatibles con *tensorflow*, no se incorporó directamente en el *pipeline*, sino que se aplicó previamente generando nuevas carpetas con audios ya procesados. Al oír los audios procesados, en los casos como el audio ruidoso de la mujer y la bebé se reducen las pausas entre su conversación pasando de 15 a 12 segundos.

La función `largo_fijo` ajusta todos los audios a una duración fija. Para ello se utilizan dos estrategias diferentes. Para audios más cortos se multiplica y concatena el audio original hasta alcanzar la duración deseada. En cambio, para audios más largos se toma de forma aleatoria 3 ventanas de largo fijo y se comparan sus energías devolviendo aquella con mayor energía. Esta función forma parte del *pipeline* y se aplica sobre los audios luego de la eliminación de silencios. Esta función tiene cierto grado de aleatoriedad que permite entrenar con distintas ventanas de audio en diferentes épocas, logrando así captar el contenido total representativo del audio y siendo un híbrido entre la estrategia de ventana aleatoria y la de tomar la ventana con mayor energía.

La otra función que se realizó dentro del *pipeline* de preprocesamiento fue la función `audio_a_imagen`, adaptada de [2], la cual genera los espectrogramas log mel a partir de los audios de largo fijo. Las imágenes resultantes tienen todas las mismas dimensiones, quedando listas para ser usadas como entradas en la red convolucional. Para esto debimos agregar que los valores pixel estén entre 0 y 255 como esperaríamos en una imagen, esto es importante especialmente para el uso de *transfer learning* donde las redes preentrenadas que usaremos asumen imágenes con valor pixel estándar.

Debido al elevado tamaño de las carpetas de los archivos de audio originales, se optó por reducir la frecuencia de muestreo de 44.1 kHz a 16 kHz y se aplicó `sacar_silencios` sobre estos .wav.

4. Métrica de evaluación - LWLRAP

La métrica utilizada para evaluar el desempeño de los modelos es *Label Weighted Label Ranking Average Precision* (LWLRAP). Esta métrica toma valores entre 0 y 1, donde valores más altos indican un mejor rendimiento. La métrica se basa en hallar el ranking relativo en lugar de un promedio simple por lo que tiene en cuenta las frecuencias relativas de las etiquetas. Esto garantiza que las etiquetas que aparecen con menos frecuencia no tengan una ventaja indebida y que se asignen las ponderaciones adecuadas a todas las etiquetas en función de su frecuencia de aparición [1]. Para el proyecto utilizamos una implementación de la métrica compatible con *TensorFlow* obtenida de [10].

5. Conjuntos de entrenamiento, validación y prueba

Como el conjunto de *test* está compuesto únicamente por datos curados, consideramos relevante que el conjunto de validación también esté conformado unicamente por este tipo de datos. Sin embargo, los audios curados son muy útiles para el entrenamiento, ya que, al ser más confiables, permiten que la red aprenda de forma más efectiva. De hecho, en las primeras pruebas con el modelo sencillo (ver subsección 6.1), observamos que entrenar únicamente con audios curados produjo mejores resultados que usar audios ruidosos, e incluso mejores que usar ambos tipos de datos combinados. Por lo que utilizaremos el 80 % de los datos curados para entrenar y el 20 % para validar.

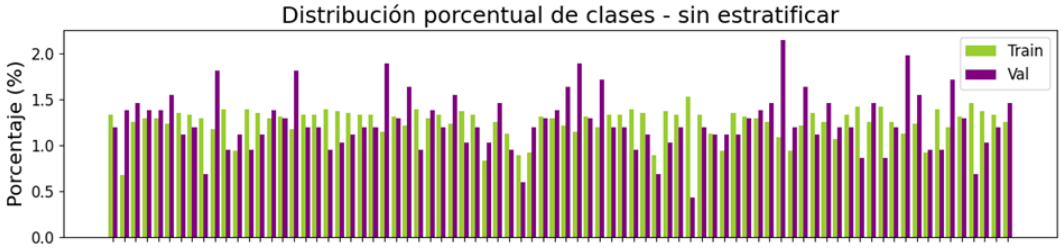

Figura 5.1: Distribución de etiquetas luego de aplicar `train_test_split` sobre el conjunto de audios curados.

Al dividir los datos curados en una proporción 80/20 mediante un `train_test_split` común, observamos que la distribución por clase quedaba desbalanceada, como se muestra en la Figura 5.1. Esto se debe a que, como se pudo ver en la Figura 3.2, los datos curados presentan un cierto desbalance de clases. Para resolver esto, utilizamos `iterative_train_test_split` [9], que permite preservar la proporción multietiqueta entre los subconjuntos de entrenamiento y validación, obteniendo así conjuntos más equilibrados. Se observa que finalmente la división es 79/21 conservando una distribución pareja por etiqueta como se ve en la Figura 5.2.

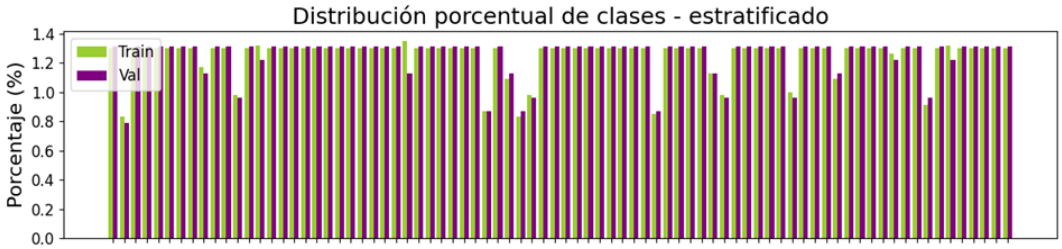

Figura 5.2: Distribución de etiquetas luego de aplicar `iterative_train_test_split` sobre el conjunto de audios curados.

Cabe aclarar que en algunos casos, cuando usamos audios ruidosos para reentrenar arquitecturas, utilizamos una división clásica 80 %, *train* y 20 % val, con `train_test_split` pues estos no presentan desbalance alguno. En este caso, el conjunto de validación se usa con el objetivo de monitorear el ajuste de los pesos durante el entrenamiento utilizando ese tipo de datos, pero no es usado para realizar una validación definitiva.

6. Modelos intermedios

El tamaño de las imágenes con las que vamos a trabajar es de 80 columnas[1] y 92 filas. Estas 92 filas responden a un audio de largo fijo 3 segundos, elegimos esta duración por ser la que generaba una relación más pareja (cuadrada) entre filas y columnas. Siendo que no era ni demasiado grande ni demasiado pequeña.

[1] parámetro `num_mel_bins`

Como función de *loss* utilizamos `binary_crossentropy` ya que estamos haciendo *multilabel binary classification* [3], y la probabilidad de pertenecer a cada clase es independiente. Es como si fueran 80 problemas binarios, donde se encuentra la probabilidad de que cierto audio contenga determinada etiqueta.

El optimizador utilizado fue 'Adam' para tener una convergencia más veloz dado que por el volumen de datos y la complejidad de los modelos el entrenamiento de cada prueba llevaba varios minutos. De el valor de la tasa de aprendizaje se hablará en cada sección ya que fue un parámetro que variamos bastante.

6.1. Modelo convolucional sencillo

Nuestro modelo básico está contenido en la función `make_model`. La entrada es normalizada entre 0 y 1 asumiendo recibe valores entre 0 y 255. Cuenta con dos bloques conteniendo capas convolucionales con activación `relu` y `padding` same para conservar relación espacial. Cada uno de estos bloques cuanta con *batch normalization* antes de la convolucional y *pooling* luego. Con estas capas se logró estabilizar el entrenamiento y evitar sobreajuste. Posteriormente una capa `GlobalAveragePooling2D`, `BatchNormalization` una capa densa de activación `relu` y una capa de *dropout*. Al final una capa densa de 80 neuronas con activación `sigmoide` para representar las probabilidades de cada clase. Si bien este modelo es simple, resultó ser efectivo aplicado sobre los audios curados, obteniendo en validación un LWLRAP de 0.45 al cabo de 50 épocas. Se utilizó optimización `Adam` con un *learning rate* de 3e-4.

6.2. Uso aumentado de datos

Tras observar sobreajuste, y dado que contamos con pocos audios curados en relación a ruidosos, entendimos conveniente realizar aumentado de datos para permitirle al modelo seguir aprendiendo al tener más datos curados. Cabe destacar que para el uso de este método se usaron solamente audios curados del conjunto de *train* y que además para ello igualmente se conservaron los audios originales. Es decir, se "duplicó" el conjunto de *train* curados aplicandosele a una mitad de ellos el aumentado y al resto ninguna transformación.

Se probaron dos diferentes tipos de aumentado de datos. Por un lado probamos una técnica de aumentado de datos llamada `MixUp` [7], que combina aleatoriamente pares de datos y sus etiquetas. Para cada par, interpola linealmente las entradas y las etiquetas usando un valor controlado por el parámetro *lambda* como se ve a continuación:

$$\widetilde{x} = \lambda x_i + (1 - \lambda) x_j$$

$$\widetilde{y} = \lambda y_i + (1 - \lambda) y_j$$

Esto genera datos intermedios que ayudan a regularizar el modelo y mejorar su generalización, especialmente en tareas con clases superpuestas. Es una estrategia interesante que crea datos mezclados combinando también las etiquetas.

Por otro lado, encontramos un *paper*, [5], que habla específicamente de aumentado de datos de espectrogramas realizando 3 tipos de transformaciones. Por un lado *Time Warping*, que deforma el espectrograma a lo largo del eje temporal simulando cambios en la velocidad del audio. La segunda es *Frequency Masking*, que enmascara aleatoriamente bandas de frecuencia, simulando pérdidas de información, como podría suceder por ruido, y *Time Masking*, que oculta intervalos de tiempo completos, representando pausas. Estas transformaciones buscan ampliar la cantidad de imágenes de espectogramas presentándole nuevos datos al modelo que podrían ser obtenidos de audios sutilmente diferentes a los del conjunto curado. Esta implementación no fue realizada por nosotros sino que adaptamos levemente un código [4] basado en este *paper* donde se implementa la función `SpecAugment`.

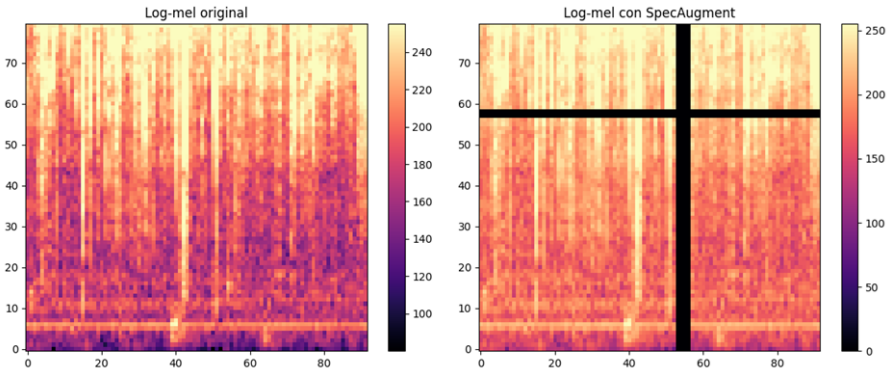

Figura 6.1: Visualización del espectro logmel de un audio curado sin aplicarle aumentado de datos y con aumentado de datos `SpecAugment`.

6.3. Utilizando ResNet50 y MobileNet preentrenadas

Para este primer entrenamiento, congelamos los pesos de la ResNet y MobileNet para mantener las representaciones aprendidas y agregamos nuevas capas convolucionales y densas al final del modelo. Para entrenar utilizamos todos los datos ruidosos y el 80 % de los datos curados y el restante 20 % se usó para validación

Se entrenaron los tres modelos durante quince épocas, el Modelo convolucional sencillo, ResNet50 y MobileNetV2. Los resultados se muestran en la Figura 6.2.

También se compararon la cantidad de parámetros utilizados por cada modelo y el tiempo que conllevó su entrenamiento en la Tabla 6.1.

En la Figura 6.2 se observa que los modelos preentrenados presentan sub ajuste, ya que a lo largo de las 15 épocas no logran mejorar significativamente. La métrica LWLRAP en entrenamiento tampoco muestra una tendencia creciente y se mantiene cercano a los valores de LWLRAP de validación, ambos en niveles bajos. En comparación, el modelo sencillo que tiene más capas convolucionales entrenables logra un mayor desempeño.

Modelo	#Parámetros	Entrenables	No entrenables	Tiempo (s)	Metrica Train	Metrica Val
Conv. Simple	37,844	37,650	194	1860	0.2896	0.2760
ResNet	24,132,816	545,104	23,587,712	2700	0.1418	0.1344
MobileNet	2,606,480	348,496	2,257,984	2400	0.1318	0.1203

Tabla 6.1: Parámetros, tiempo de entrenamiento y métrica LWLRAP de cada modelo.

Se evaluó unir la convolucional sencilla a la salida de ResNet y MobileNet, lo que mostró una mejoría respecto al uso de una cabeza solamente de capas densas como se puede ver en la Tabla 6.2. En comparación con la Tabla 6.1, vemos que conviene entrenar capas convoluciones con espectrogramas para abordar mejor nuestro problema, es por esto que en la siguientes secciones habilitamos capas de las arquitecturas vistas. Se aclara que los tiempos de ejecución son comparables dentro de una misma tabla únicamente, ya que fueron ejecutados en diferentes dispositivos.

Modelo	LWLRAP *train*	LWLRAP val	Tiempo (s)
ResNet	0.5388	0.3499	1427
MobileNet	0.4751	0.3259	761

Tabla 6.2: Comparación luego de entrenamiento durante 15 épocas sobre el conjunto de datos curados y ruidosos para *transfer learning* agregando el modelo convolucional simple a la salida de cada una de las arquitecturas.

Para el siguiente experimento, se mantuvieron congeladas las capas de MobileNet y ResNet, y se aplicó un aumento de datos sobre el conjunto de entrenamiento (SpecAugment).

Estos cambios resultaron en una mejoría significativa en el desempeño de los tres modelos, como se muestra en la Figura 6.3.

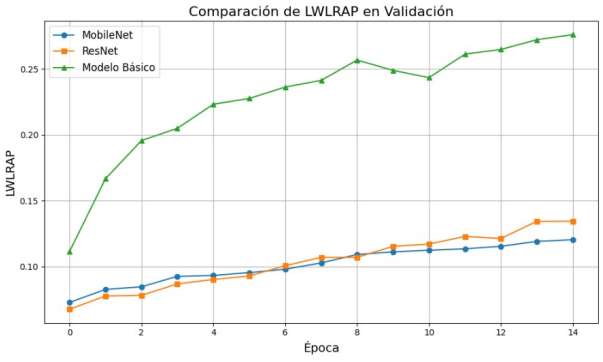

Figura 6.2: Comparación de modelos sin aumento de datos.

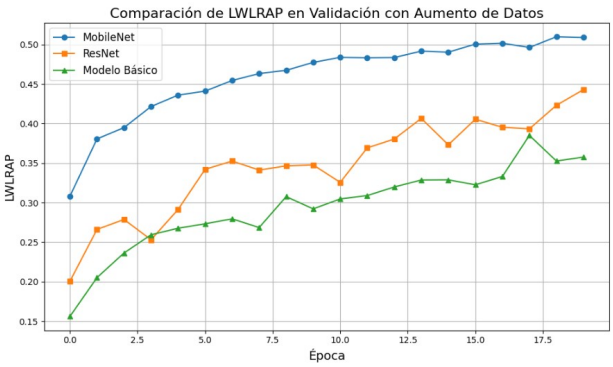

Figura 6.3: Comparación de modelos con aumento de datos con `SpecAugment`.

6.4. Utilizando ResNet50 y MobileNet entrenando últimas capas

Se utilizaron las arquitecturas ResNet50 y MobileNet y se levantaron capas a entrenar con los ejemplos ruidosos para modificar parte de los parámetros aprendidos por estas arquitecturas. Para esto se utilizaron los datos ruidosos para entrenar solo las partes finales de cada una de las redes, ya que las CNNs aprenden más los detalles en los filtros finales. En esta sección se consideraron las arquitecturas ResNet50 entrenando los bloques 4 y 5 y MobileNet entrenando las etapas 11 a 17 con los datos ruidosos. A la salida de las mismas se agregó como cabeza el modelo

sencillo descrito en 6.1, ya que en la sección anterior demostró mejor desempeño que una cabeza densa simple. Luego del primer entrenamiento con ruidosos se fijaron los pesos de ambas arquitecturas (a excepción de la cabeza) y se volvió a entrenar solo con curados. Esto permitió aprovechar la gran cantidad de datos disponibles en ese subconjunto, a pesar de que las etiquetas no fueran del todo confiables. El objetivo fue que la red aprendiera representaciones generales sobre los espectrogramas y utilizara todos los datos disponibles.

lr	LWLRAP *train*	LWLRAP val
1e-2	0.3672	0.3197
1e-3	0.4235	0.3724
1e-5	0.1115	0.1882

Tabla 6.3: Comparación de *learning rate* en ResNet50 con los últimos dos bloques entrenables durante 15 épocas sobre el conjunto de los ruidosos.

Para el primer entrenamiento (con datos ruidosos) de los dos modelos, se utilizó el optimizador Adam y se evaluaron los 3 valores de *learning rate* que se ven en la tabla 6.3. Se puede ver que el mejor desempeño se obtiene para lr=1e-3 por lo que tanto para esta arquitectura como para MobileNet usamos este valor. Con una tasa mayor oscilaba entre varios valores y con una tasa menor no conseguía aprender.

Modelo	Capas entrenables	Tiempo (s)	LWLRAP *train* ruidosos	LWLRAP val ruidosos
ResNet50	bloque 5 y 4	1875	0.4235	0.3724
MobileNet	etapas 11-17	825	0.4167	0.3686

Tabla 6.4: Comparación de los modelos realizando entrenamiento de la primer etapa entrenando últimas capas de cada arquitectura sobre el conjunto ruidoso durante 15 épocas.

Para las segundas partes del entrenamiento, es decir fijando ahora ambas arquitecturas y reentrenando sobre los datos curados, se evaluó el desmepeño sin y con los aumentado de datos descritos en 6.2. Se utilizó optimizador Adam y una tasa de aprendizaje de 3e-4.

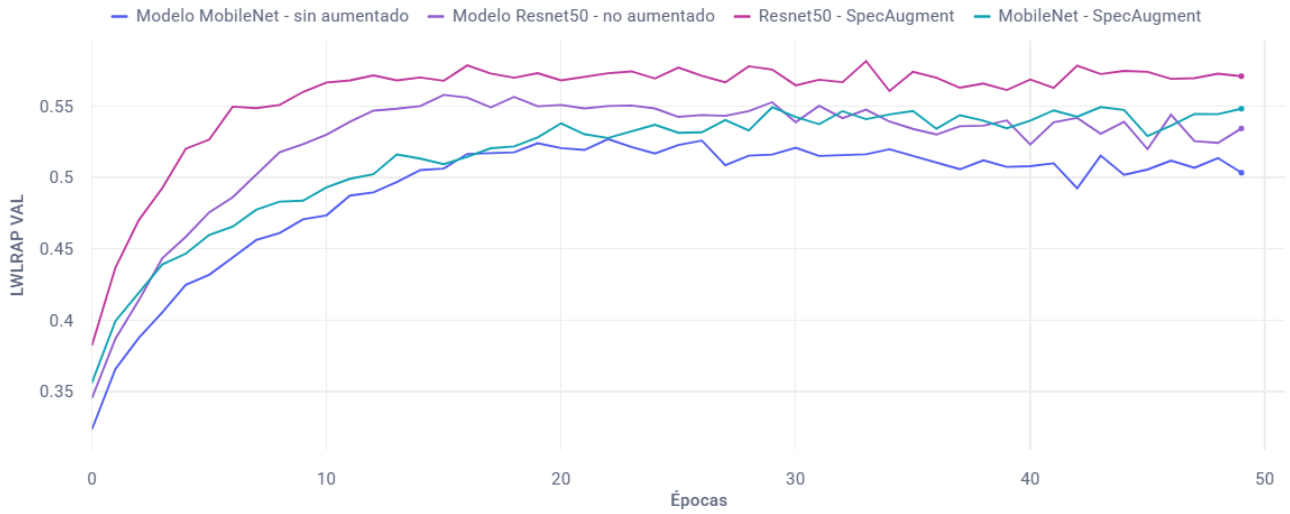

Figura 6.4: 50 épocas del segundo entrenamiento de ResNet y MobileNet con datos curados crudos y duplicando el conjunto de curados aplicandole a una de las partes `SpecAugment`.

Para las segundas partes del entrenamiento, es decir fijando ahora ambas arquitecturas y reentrenando sobre los datos curados, se evaluó el desmepeño sin y con los aumentados de datos descritos en 6.2. Se utilizó optimizador Adam y una tasa de aprendizaje de 3e-4. En la tabla 6.5 se encuentran los valores de la métrica para cada uno de los casos (`SpecAugment` y `MixUp`) y en la figura 6.4 la comparación utilizando `SpecAugment` y no realizando aumentado de datos.

Intentamos unir los dos tipos de aumentados de datos pero no nos fue posible por lo que descartamos el uso del MixUp por alcanzar mejores desempeños con SepcAugment. Quizás con los dos tipos de aumentados podríamos haber alzanzado mejor desempeño que el SepcAugment por sí solo ya que con este aumentado se visualiza cierto sobreajuste y el aumentado de datos es una forma de regularización. Podemos ver que luego de 50 épocas el entrenamiento es bueno pues hay un gran sobreajuste debido a la cantidad de épocas evaluadas, esto no sucedió así con MixUp.

	Sin aumentado de datos		SpecAugment		MixUp	
Modelo	LWLRAP *train* curados	LWLRAP val curados	LWLRAP *train* curados	LWLRAP val curados	LWLRAP *train* curados	LWLRAP val curados
ResNet	0.9690	0.5344	0.8277	0.5750	0.4615	0.5012
MobileNet	0.9366	0.5034	0.7246	0.5482	0.4154	0.4582

Tabla 6.5: Comparación de desempeños durante segunda parte del entrenamiento durante 50 épocas con datos curados sin realizar aumentado y realizando aumentado `SpecAugment` y `MixUp`.

Vemos que el mejor modelo de esta sección es ResNet aplicando `SpecAugment`, lo llamaremos **modelo 2**.

6.5. Utilizando ResNet50 y MobileNet entrenando todas las capas

Luego de observar una mejora significativa en el desempeño gracias al aumento de datos y a levantar capas de las arquitecturas, se procedió a entrenar por completo ambas arquitecturas solo con nuestros datos.

El desempeño de entrenar por completo las dos arquitecturas se puede ver en la Figura 6.5 donde se realizó aumentado de datos. El modelo que dió mejor en este caso fue MobileNet, a este lo llamaremos **modelo 1**.

6.6. Fine-tunning

Con el objetivo de seguir afinando los modelos, se procedió a ajustar algunos hiperparámetros clave.

En particular, se exploró el impacto del tamaño del *kernel* en el desempeño del modelo convolucional sencillo. La Figura 6.6 muestra un ejemplo de esta experimentación, donde se comparan distintos valores de *kernel size* y su efecto sobre el rendimiento del modelo. Obteniendo como mejor tamaño de *kernel* 3×3 que fue el utilizado en todas las pruebas previas.

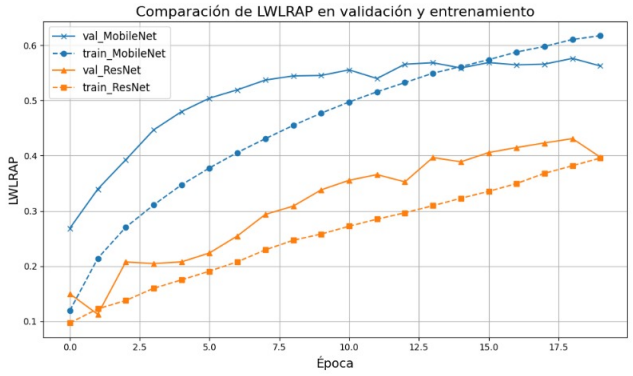

Figura 6.5: Comparación de modelos analizados en sección 6.5 utilizando la cabeza convolucional sencilla y aplicando `SpecAugment`.

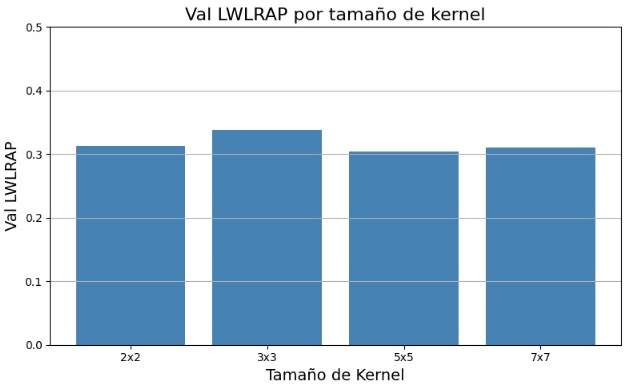

Figura 6.6: Comparación del *kernel size* en la convolucional sencilla.

Para el modelo 2, se observa que la mejor tasa de aprendizaje es la utilizada previamente, 1e-3. Tasas mayores conducen a un fuerte sobreajuste, lo cual se evidencia con un muy alto desempeño en *train* que no es acompañado en validación. Por otro lado, tasas menores no permiten al modelo aprender lo suficiente, como se puede ver en Tabla 6.6 se obtiene un valor en entrenamiento muy bajo.

optimizador	**Adam**		**SGD**	
lr	**LWLRAP *train***	**LWLRAP val**	**LWLRAP *train***	**LWLRAP val**
1e-3	0.9068	0.5882	0.2428	0.2676
3e-4	0.8277	0.5750	0.2409	0.2677
1e-5	0.3379	0.3607	0.2429	0.2657

Tabla 6.6: Comparación de *learning rate* en ResNet50 y cabeza convolucional sencilla entrenado en la segunda etapa sobre datos curados durante 20 épocas con optimizador Adam y SGD.

También confirmamos que el optimizador Adam es más adecuado que SGD para este problema. El uso de SGD no solo conduce a un rendimiento significativamente menor, sino que incluso implica un retroceso respecto al aprendizaje obtenido en la primera etapa con datos ruidosos. A pesar de probar con diferentes tasas de aprendizaje, los valores

de LWLRAP obtenidos con SGD se mantienen bajos y casi constantes, por lo que el modelo no logra aprovechar la información de los datos curados.

Para el modelo 1, se probaron diferentes valores de *learning rate* y distintos optimizadores. Al utilizar un *learning rate* de 1×10^{-3} en lugar de 3×10^{-4}, se obtuvo un valor de *LWLRAP* de 0,5020 en validación. Por otro lado, al cambiar el optimizador de Adam a SGD, el modelo no logró aprender, alcanzando un desempeño de solo 0,0656 en validación.

7. Modelo final

Sobre el conjunto de *test* resampleado a 16k Hz aplicamos la función `sacar_silencios`. Posteriormente se le realizó el mismo preprocesamiento que a los datos de entrenamiento, para un caso el preprocesamiento para pasar por una ResNet y en el otro el correspondiente para ingresar a una MobileNet, invocando nuestras funciones explicadas en 3.

Para el modelo 1, se entrenaron todas las capas como se describe en la Sección6.5, dejando un 20 % de los datos curados para validación y el 80 % restante para entrenamiento, junto con los datos ruidosos. Al 80 % de los datos curados utilizados para el entrenamiento se les aplicó `SpecAugment`. El desempeño del modelo se presenta en la Tabla 7.1.

Para evaluar en test el modelo final en el caso semientrenado de Resnet50 con la convolucional sencilla a la cabeza, el entrenamiento se realizó sobre el 90 % de los datos, reservando un 10 % para *callbacks* que evalúa el mejor modelo de momento en validación. Aplicándole aumentado de datos `SpecAugment` sobre el conjunto de los curados. Finalmente se realizó *predict* obteniendo el resultado que figura en la Tabla 7.1 como resultado de la *submission* en Kaggle.

Referencia	**Modelo**	**LWLRAP *test***
1	MobileNet completamente entrenado, con `SpecAugment`	0.43131
2	ResNet50 semi entrenado, con `SpecAugment`	0.43470
3	`MixUp`	0.43859
4	Combinación de 1 y 2	0.47128
5	Combinación de 1, 2 y 3	0.47128

Tabla 7.1: Comparación de desempeño en el conjunto de *test* de los mejores modelos.

En la Tabla 7.1 se observa que la combinación de ambos modelos, mediante promedio simple, supera el rendimiento individual. Esto muestra que la combinación de arquitecturas permite aprovechar las fortalezas de cada una y mejorar la generalización en *test*.

7.1. Modelo 3 *(MixUp)*

Con el objetivo de mejorar el desempeño, se intentó cambiar la cabeza de la red por una mas sencilla compuesta por dos capas densas ocultas de 512 y 256 neuronas respectivamente, utilizando *DropOut* como técnica de regularización. Para probar el desempeño de este modelo se escogió *MixUp* como técnica de aumentado para los datos curados.
Se repitió el mismo experimento con este modelo y se obtuvieron los resultados que se observan en la Tabla 7.1, tanto para el Modelo individualmente como para la combinación con los Modelos 1 y 2.
Se esperaba que una cabeza más sencilla

8. Conclusiones

Destacamos el abordaje del problema de audio como imágenes que nos permitió el uso de *transfer learning* con arquitecturas preentrenadas como ResNet50 y MobileNet, donde pudimos reutilizar representaciones aprendidas sobre imágenes generales y adaptarlas al dominio de los espectrogramas de diferentes formas. También el uso de *data augmentation* que fue importante para ayudar a regularizar los modelos.

En relación al uso de datos ruidosos, si bien poseen etiquetas menos confiables, su volumen es de gran valor ya que nos permitió trabajar con arquitecturas complejas que lograron ajustarse a espectogramas.

Finalmente, si bien se alcanzó un LWLRAP menor en test que en validación, se logró una generalización razonable considerando la complejidad del problema y la limitada cantidad de datos curados disponibles.

Referencias

[1] Multi-label audio classification — Freesound Audio Tagging 2019 Competition. https://medium.com/@forsomethingnewsid/multi-label-audio-classification-freesound-audio-tagging-2019-competition-7780e54 2023.

[2] Instituto de Ingeniería Eléctrica Facultad de Ingeniería. Taller de Aprendizaje Automático. Segundo Proyecto - Procesamiento de Señales de Audio. https://eva.fing.edu.uy/pluginfile.php/549401/mod_resource/content/1/TAA___Slides___Procesamiento_de_Audio.pdf, 2025.

[3] Aurélien Géron. *Hands-on Machine Learning with Scikit-learn, Keras & TensorFlow*. 3a ed. O'Reilly, 2022.

[4] MichaelisTrofficus. Implementation of SpecAugment, a simple data augmentation technique for speech recognition. https://github.com/MichaelisTrofficus/spec_augment, 2022.

[5] Daniel S. Park, William Chan, Yu Zhang, Chung-Cheng Chiu, Barret Zoph, Ekin D. Cubuk, and Quoc V. Le. Specaugment: A simple data augmentation method for automatic speech recognition. *arXiv preprint arXiv:1904.08779*, 2019. https://arxiv.org/abs/1904.08779.

[6] Onkar Patil. How to Remove Silence from an Audio using Python. https://onkar-patil.medium.com/how-to-remove-silence-from-an-audio-using-python-50fd2c00557d, 2022.

[7] Sayak Paul. MixUp augmentation for image classification. https://keras.io/examples/vision/mixup/, 2024.

[8] Vaibhav Singh. Enhancing Medical Multi-Label Image Classification Using PyTorch & Lightning. https://learnopencv.com/medical-multi-label/, 2023.

[9] Paweł Szymański and Tomasz Kajdanowicz. A scikit-based python environment for performing multi-label classification. *arXiv preprint arXiv:1702.01460*, 2017. http://scikit.ml/stratification.html.

[10] Carl Thomé. lωlrap metric for tf.keras. https://www.kaggle.com/code/carlthome/l-lrap-metric-for-tf-keras, 2021.