Universidad Austral de Chile

Conocimiento y Naturaleza

SONIDOS DE OFICINA SINTÉTICOS: DETECTANDO EVENTOS EN UN ENTORNO VIRTUAL

Proyecto presentado como parte de los requisitos del ramo

INFO343: Minería y Aprendizaje con Datos

Escuela de Graduados

Facultad de Ciencias de la Ingeniería

Universidad Austral de Chile

CRISTIAN ANTONIO COFRÉ SEPÚLVEDA

VALDIVIA-CHILE

2023

Introducción

En el mundo actual, la inteligencia artificial y el procesamiento de señales desempeñan un papel crucial en una amplia gama de aplicaciones. Uno de los campos en constante evolución es la detección de eventos de sonidos en entornos diversos. En este contexto, se presenta un emocionante desafío: entrenar una red neuronal para la detección de eventos de sonidos en mezclas sintéticas, específicamente aquellos que ocurren en un entorno de oficina.

Este desafío tiene como objetivo abordar un escenario realista pero controlado: la identificación y separación de eventos de sonidos superpuestos en una oficina simulada. Estos eventos incluyen sonidos comunes como toses, risas, timbres de teléfono y más.

Descripción de set de datos

El conjunto de datos para este desafío consta de dos componentes principales: eventos de sonidos aislados para cada categoría, llamado conjunto de datos de entrenamiento (que denominaremos como conjunto aislado), y mezclas sintéticas que contienen ejemplos de sonidos en diversas condiciones, construidas utilizando las muestras del conjunto de datos aislado, llamado conjunto de datos de desarrollo (que denominaremos como conjunto mixto). Las categorías de eventos de sonidos, en ambos conjuntos, incluyen once tipos distintos, como:

- clearthroat: Aclarar la garganta
- cough: Tos
- doorslam: Portazo (portazo de puerta)
- drawer: Cajón

- keyboard: Teclado (clics de teclado)
- keys: Caída de llaves (llaves caídas en el escritorio)
- knock: Golpear (golpeando en la puerta)
- laughter: Risa
- pageturn: Pasar página (pasar de página de papel)
- phone: Teléfono (timbre de teléfono vintage)
- speech: Discurso

Cada categoría cuenta con 20 ejemplos en el conjunto de aislado. Además, el conjunto mixto contiene 18 minutos de mezclas sintéticas en archivos de audios de 2 minutos de duración.

Los archivos de audio utilizados en el conjunto aislado fueron proporcionados por IRCCYN, École Centrale de Nantes. La grabación se realizó en un entorno tranquilo utilizando un micrófono de cañón AT8035 conectado a un grabador ZOOM H4n. Los archivos de audio tienen una frecuencia de muestreo de 44.1 kHz y son monofónicos.

El conjunto de datos aislado está compuesto por una serie de archivos de audio en formato "wav", donde la etiqueta correspondiente a cada audio se refleja en el nombre del archivo.

En lo que respecta a los datos mixtos, la situación es un poco más compleja debido a que son audios simulados. Esto implica la presencia de una serie de parámetros de simulación que se detallan en los nombres de los archivos de audio, el la carpeta "sound". Estos parámetros son:

- ebr: Relación entre Eventos y Fondo (Event to Background Ratio).
- nec: Número de Eventos Activos por Clase (Number of active Events per Class).

- poly: Valor booleano que indica si la escena es polifónica (poly=1; los eventos pueden superponerse), o monofónica (poly=0; los eventos no pueden superponerse).

Las etiquetas de los audios simulados se encuentran en la carpeta “annotation”, donde se ubican archivos con extensión “text”. Cada archivo tiene el mismo nombre que los archivos de audio correspondientes. Dentro de estos archivos, la primera columna representa un intervalo de tiempo y la segunda columna indica la etiqueta asociada a ese intervalo.

Objetivos

- Evaluar el rendimiento de la red Neuronal entrenada con el conjunto de datos aislados.
- Evaluar el rendimiento de la red Neuronal entrenada con el conjunto de datos mixtos.
- Entrenar la red neuronal utilizando datos aislados y mixtos.
- Comparar el rendimiento de las redes neuronales entrenadas y determinar la más efectiva para predecir el conjunto de prueba.

Preprocesamiento de datos

El siguiente procedimiento está contenido en el archivo ”procesamiento.py”. Para adecuar los datos a un formato legible por la red neuronal, fue esencial realizar un proceso de preprocesamiento. Cabe destacar que este procedimiento difiere entre los dos conjuntos de datos debido a sus características particulares. En el caso del conjunto de datos aislados, se empleó la función “extract_label_from_filename” para

extraer la etiqueta del nombre de los archivos de audio, junto con el nombre completo del respectivo audio. Estos resultados se guardaron en formato CSV en la carpeta "meta".

En relación al conjunto de datos mixto, el proceso se volvió un tanto más tedioso debido a la presencia de audios polifónicos. Para este proyecto, se optó por utilizar únicamente audios no polifónicos, es decir, aquellos que contienen solo una etiqueta por audio. Afortunadamente, como mencionamos previamente, la información sobre si un audio es polifónico o no está implícita en su nombre. Un segundo desafío en este conjunto surge debido a que, incluso al descartar los audios polifónicos, los archivos contienen más de una etiqueta, ya que incluyen distintos sonidos aunque no estén superpuestos.

Para resolver esta situación, se llevó a cabo la división de los audios en segmentos según los detalles proporcionados en los archivos ubicados en la carpeta .ªnnotation", mencionada previamente. Esta tarea fue realizada mediante la función "separate_audio_with_txt", lo que permitió segmentar los audios en diferentes secciones y guardar tanto el nombre del audio como su etiqueta correspondiente en la carpeta "meta".

Finamente se utilizó el código "dataset.py", para preparar y gestionar datos de audio de manera eficiente en el entrenamiento y evaluación del modelo. Primero, se lee un archivo CSV de metadatos ubicado en la subcarpeta "meta"del directorio principal, que contiene información sobre los archivos de audio y sus etiquetas.

Se inicializa un codificador de etiquetas (LabelEncoder) para asignar etiquetas de texto a valores numéricos. Las formas de onda de audio se cargan desde las rutas de archivo, y se aplican transformaciones, en este caso será el espectrograma mel (detallada en la siguiente sección), a las formas de onda.

Posteriormente se retorna una tupla que contiene la forma de onda de audio transformada, la etiqueta numérica y la ruta del archivo.

Transformación del Audio

Para facilitar la extracción de características significativas a partir de los datos de entrada de audio, se requiere llevar a cabo un proceso de procesamiento de señales de audio. El objetivo de este proceso es generar el espectrograma mel de cada audio, lo que permitirá crear representación que resalta las características esenciales, elimina detalles menos relevantes, y ser utilizada como entrada en una red neuronal.

Para obtener la entrada se utilizó la clase llamada "featurizer_pipeline", dentro de "train.py". La clase inicializa parámetros importantes, para generar el espectrograma mel, como la tasa de muestreo final (sample_rate), la longitud de la ventana de análisis (win_length) y el número de coeficientes mel (n_mels). Estos parámetros son proporcionados en el diccionario params dentro de param.yaml.

Luego, se crea una instancia del transformador Resample. Este transformador se utilizará para ajustar la frecuencia de muestreo de los datos de entrada de audio a la tasa de muestreo final.

Posteriormente, se instancia un transformador MelSpectrogram llamado mel_transform. Este transformador se utiliza para calcular el espectrograma mel de los datos de entrada. El espectrograma mel se calcula utilizando la nueva tasa de muestreo, una ventana de análisis con longitud win_length y se divide en n_mels bandas mel.

Una vez obtenida el nuestro dataset con sus respectivas transformaciones, vemos que existen audios que tienen diferentes duraciones de tiempo. Para solucionar este problema utilizamos "data padding" para manejar esta variabilidad temporal. Lo

que implica agregar valores de relleno a los audios más cortos para que tengan la misma longitud que los más largos.

Para realizar esto utilizamos la función "padd", dentro de "train.py", que recibe un conjunto de datos con mel espectrogramas, etiquetas y archivos de audio, y se encarga de igualar la longitud de los espectrogramas mediante relleno. Calcula automáticamente la longitud máxima y el valor mínimo en los espectrogramas si no se proporcionan, luego agrega relleno a cada espectrograma para que tengan la misma longitud máxima. Los datos rellenados se almacenan junto con las etiquetas y archivos de audio en una lista de tripletes, que luego se devuelve como resultado. Este dataset generado será utilizado para el entrenamiento y evaluación de la red neuronal.

Modelo de Red Neuronal

La arquitectura (ver Figura 1) comienza con capas convolucionales seguidas de operaciones de normalización y activaciones ReLU. Estas capas convolucionales (con tamaños de filtro 5x5 y padding de 2) detectan patrones y características en los espectrogramas Mel. Las capas de MaxPooling (con tamaño de ventana 2x2) reducen la dimensionalidad. Posterior se introduce Dropout, para reducir el riesgo de sobreajuste. El hiperparámetro drop_rate controla la cantidad de unidades que se desactivan aleatoriamente durante el entrenamiento, este se define en el diccionario params.

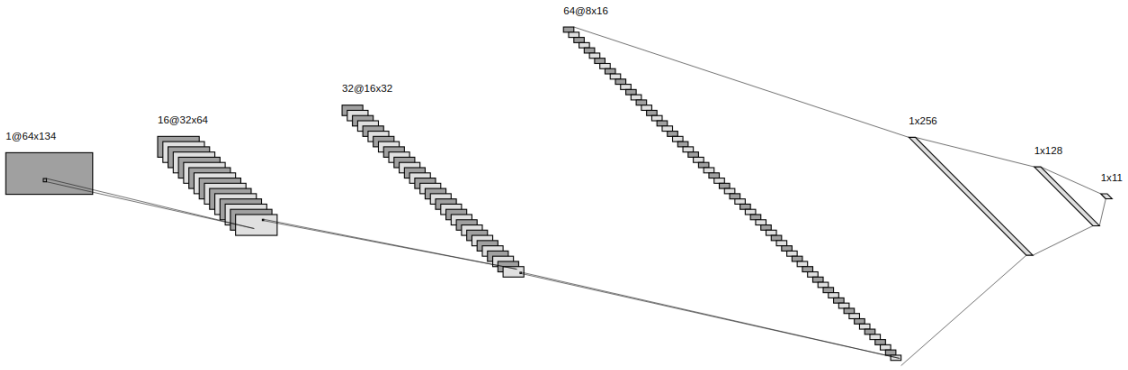

Figura 1: Arquitectura del Modelo de Red Neuronal Convolucional (CNN) para la Clasificación de Etiquetas en Sonidos con Transformación Mel

Después de las capas convolucionales, las características se aplanan y se pasan a través de capas completamente conectadas. Estas capas aprenden combinaciones lineales de las características extraídas por las capas convolucionales. Entre las capas lineales, se aplican activaciones ReLU y capas Dropout.

La última capa lineal produce las predicciones de clasificación. Se utilizan 11 unidades de salida, una por cada posible etiqueta. No se aplica activación en la capa de salida ya que se utilizará la función Cross Entropy Loss.

En cuanto a la optimización de parámetros se utilizó “Adam” y la tasa de aprendizaje se define en el diccionario params.

Esta arquitectura es útil para el entrenamiento de redes neuronales en la predicción de etiquetas en sonidos debido a su capacidad para aprender características relevantes a partir de las representaciones Mel de los espectrogramas de audio. Las capas convolucionales capturan patrones y detalles locales, mientras que las capas completamente conectadas realizan combinaciones lineales más abstractas. La regularización mediante Dropout y las funciones de activación ReLU ayudan a evitar el sobreajuste y a introducir no linealidades en el modelo. La función de pérdida Cross Entropy Loss es apropiada para problemas de clasificación como este.

Entrenamiento

El siguiente procedimiento fue realizado con la librería"PyTorch" junto a "Lightning". El proceso inicia con la creación de dataloaders, donde se generan dos instancias: train_loader y valid_loader, diseñadas para cargar conjuntos de datos de entrenamiento y validación, respectivamente. En train_loader, el tamaño de lote (batch_size) se ajusta conforme al valor definido en el diccionario params, mientras que en valid_loader, se fija un tamaño de lote de 60.

A continuación, se procede con la configuración de callbacks. La utilidad de los callbacks radica en supervisar y controlar el entrenamiento. El callback ModelCheckpoint asume la responsabilidad de almacenar el mejor modelo con base en la métrica de pérdida de validación (val_loss). Por otro lado, EarlyStopping se encarga de detener el entrenamiento si la pérdida de validación no presenta mejoras durante un período de paciencia determinado, en este caso, 200 épocas.

El entrenador (trainer) se configura con distintos parámetros que influyen en la dinámica del proceso. max_epochs define el número máximo de épocas de entrenamiento, según la especificación en params. Asimismo, check_val_every_n_epoch establece la frecuencia de evaluación en los datos de validación, realizando una evaluación después de cada época. Para el registro de información relevante durante el entrenamiento, se emplean loggers como CSVLogger y TensorBoardLogger.

En conjunto, esta metodología busca optimizar la generalización del modelo, evitando el sobreentrenamiento y asegurando que el mejor desempeño sea capturado y guardado. Cada aspecto, desde la arquitectura del modelo hasta la gestión de la convergencia, se considera cuidadosamente, lo que contribuye a un proceso de entrenamiento riguroso y efectivo.

Optimización de Hiperparámetros

Antes de entrenar la red neuronal, se llevó a cabo una optimización de los hiperparámetros relacionados con la transformación del audio mencionada previamente. Para realizar este proceso, se utilizó el script "opt_hpt.py", el cual hace uso de la biblioteca "optuna". El procedimiento consistió en variar el parámetro "sample_rate" en valores entre 2000 y 8000, con un incremento de 1000 en cada iteración. Los demás hiperparámetros se detallan en la Tabla 1 (diccionario params), y se evaluó el rendimiento de cada modelo mediante la métrica de accuracy entre las predicciones y los resultados reales, utilizando el conjunto de validación. En consecuencia, el objetivo fue maximizar el valor de la accuracy para identificar el hiperparámetro óptimo.

features	
sampling_rate	opt
n_mel_coefs	64
training	
learning_rate	0.0001
drop_rate	0.5
batch_size	32
max_epochs	100
dataset_split_seed	1234

Tabla 1: Configuración de Hiperparámetros

Resultados y Discusión

El primer proceso de entrenamiento se llevó a cabo utilizando el conjunto de datos aislado, empleando los hiperparámetros detallados en la tabla 1 y configurando un valor de "sample_rate" de 6000, obtenido tras una fase de optimización previa. Para la evaluación de los resultados, se empleó un conjunto de pruebas compuesto por

datos virtuales, similares a los datos del conjunto mixto. La generación de estos datos virtuales se realizó siguiendo el mismo procedimiento de extracción empleado en el conjunto de datos mixto, mediante la ejecución del script "procesamiento.py". En este contexto, el conjunto de datos aislado consta de un total de 220 audios con 20 muestras por cada etiqueta. De estas, se emplearon 180 para llevar a cabo el proceso de entrenamiento del modelo, mientras que las restantes 40 se destinaron para validación.

Los resultados del modelo están representados en la Figura 2, que muestra la matriz de confusión. En esta matriz, se presentan las etiquetas reales y las predichas, lo que nos permite visualizar y analizar el desempeño del modelo en la clasificación.

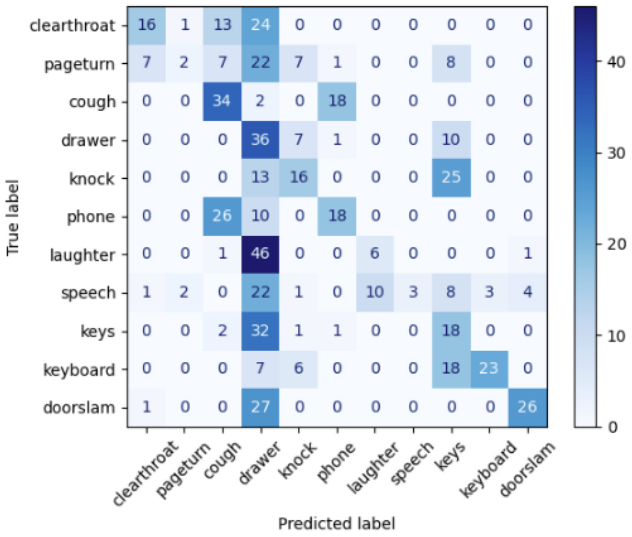

Figura 2: Matriz de Confusión de modelo con datos aislados

Para la evaluación del modelo, se realizaron cálculos adicionales además de la matriz de confusión, centrándose en métricas como precisión y recall (ver Figura 3). Estas métricas ofrecen una visión precisa de cómo el modelo lleva a cabo las clasificaciones, destacando tanto los errores de tipo I (falsos positivos) como los errores de tipo II (falsos negativos). Este enfoque proporciona una comprensión más

detallada de la capacidad del modelo para identificar correctamente las instancias positivas y negativas en la tarea de clasificación.

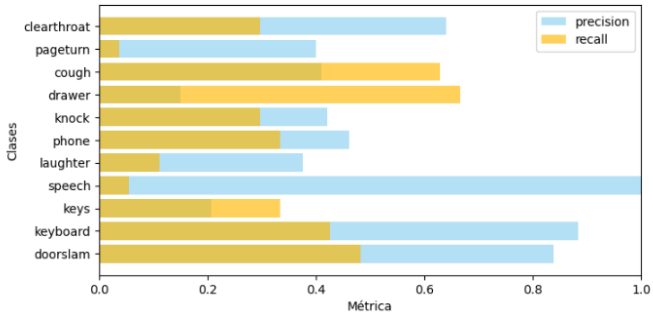

Figura 3: Métricas de testeo de modelo con datos aislados

En los resultados, podemos observar que existen etiquetas como "speech"que muestran una alta precisión pero un recall muy bajo. Esto indica que el modelo está haciendo predicciones de esta etiqueta confundiéndola con otra, en este caso, "drawer". Queda claro entonces que una alta precisión no garantiza necesariamente un buen rendimiento del modelo, resaltando la importancia de buscar altos valores en ambas métricas.

Las predicciones con mejor rendimiento en este modelo son "keyboard", "doorslamz çlearthroat", ya que muestran valores decentes en ambas métricas. A pesar de que al observar la matriz de confusión se aprecian etiquetas con valores más altos en la diagonal, como çoughz "drawer", su baja precisión indica que el modelo está realizando predicciones que se corresponden con varias etiquetas, como las mencionadas anteriormente.

En conclusión, es fundamental destacar que el valor de accuracy alcanzado por el modelo es de 0.33. El uso de la métrica de accuracy nos brinda una perspectiva global sobre el rendimiento del modelo en todas las clases. En este escenario particular, el modelo logra un accuracy del 33 %, lo que implica que aproximadamente el 33 % de las muestras son clasificadas correctamente. Este resultado subraya un nivel de

desempeño que puede considerarse bajo en términos de la capacidad general de predicción del modelo en el conjunto de datos evaluado.

En la segunda fase de entrenamiento, se procedió a entrenar el modelo utilizando el conjunto de datos mixto. De manera análoga al primer entrenamiento, se emplearon los mismos hiperparámetros definidos en la tabla 1, ajustando además el valor de "sample_rate" a 7000. La misma colección de datos de prueba utilizada en el primer entrenamiento fue empleada aquí nuevamente.

En este contexto, el conjunto de datos mixto se compone de un total de 198 registros de audio, con 18 muestras disponibles para cada etiqueta. De este conjunto, se seleccionaron 163 muestras para el proceso de entrenamiento del modelo, reservando las 35 muestras restantes para su utilización en la fase de validación del entrenamiento.

La representación de los resultados del modelo se encuentra ilustrada en la Figura 4, donde se presenta la matriz de confusión correspondiente.

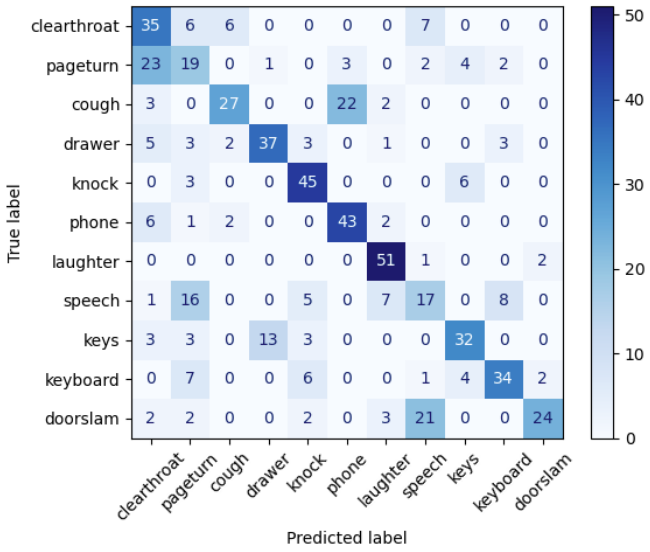

Figura 4: Matriz de Confusión de modelo con datos Mixtos

El segundo entrenamiento, realizado con el conjunto de datos mixto, exhibe

una notable mejora en los resultados, como evidencia la matriz de confusión y la representación en la Figura 5. Se observa una mayor equidad en las métricas, lo que indica que el modelo ha logrado una mayor seguridad en sus predicciones. A pesar de ello, persisten algunas etiquetas con cierta confusión, como en el caso de "doorslam", que el modelo confunde ocasionalmente con "speech". A pesar de estas instancias, el rendimiento general del modelo es satisfactorio, con un valor de precisión global de 0.61.

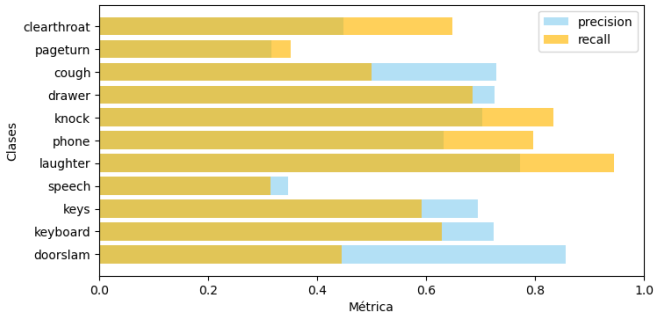

Figura 5: Métricas de testeo de modelo con datos aislados

Finalmente, procedimos a entrenar el modelo utilizando la combinación de ambos conjuntos de datos (aislado y mixto), el cual denominó modelo acoplado, conformado por un total de 418 grabaciones. En este escenario, se emplearon 334 muestras para el proceso de entrenamiento y las restantes 84 fueron destinadas para la validación del modelo. El "sample_rate" utilizado para este modelo fue 6000.

Los resultados obtenidos en este modelo muestran similitudes con los resultados anteriores. Observando la figura 6 vemos que se ha logrado resolver confusiones que se presentaban previamente, como el caso en el que el modelo clasificaba incorrectamente instancias de "doorslam" como "speech".

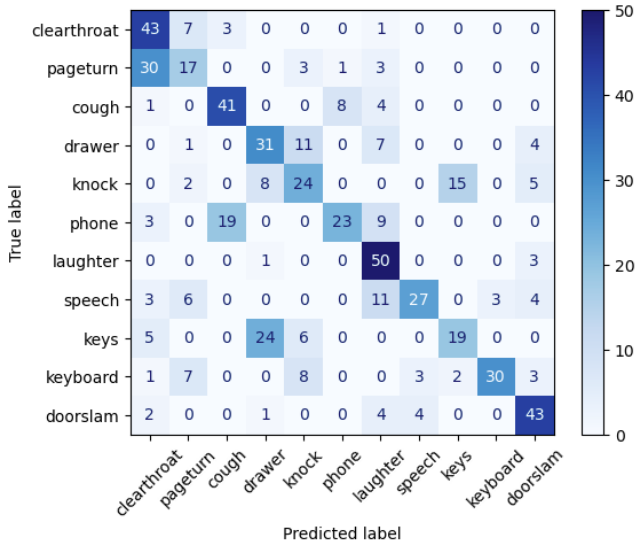

Figura 6: Matriz de Confusión de modelo con acoplado

A pesar de lo mencionado anteriormente, al analizar detenidamente la representación gráfica en la Figura 7, se puede apreciar que los resultados siguen siendo comparables a los obtenidos en el modelo con datos mixtos. Sin embargo, es importante señalar que el valor de accuracy ha disminuido a 0.59 en esta instancia. Esta disminución en la precisión general puede atribuirse a la inclusión del conjunto de datos aislado, lo cual parece haber afectado ligeramente el rendimiento del modelo en algunas clases.

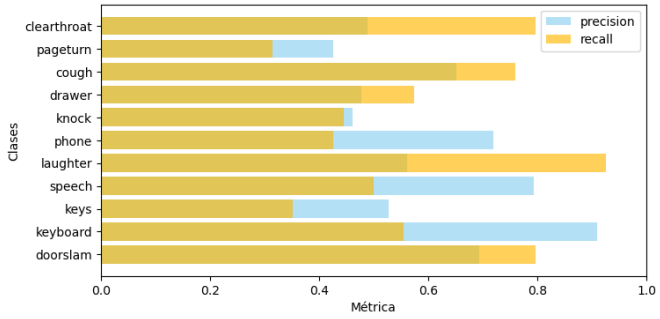

Figura 7: Métricas de testeo de modelo con acoplado