Inteligencia Artificial para la Clasificación Sonora Aplicada a Dispositivos de Indicación y Monitorización de la Contaminación Acústica

Ingeniería

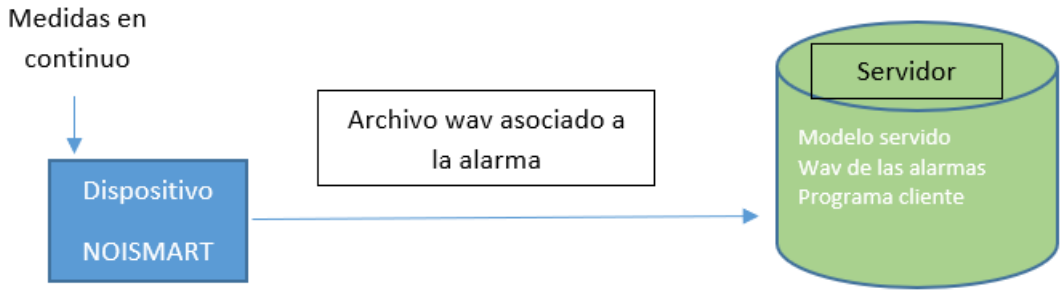
SEMÁFORO RUIDO SL Bilbao

DESCRIPCIÓN EL SISTEMA CLASIFICADOR DE SONIDOS URBANOS

1. Esquema general del sistema de clasificador de alarmas (identificador de sonidos urbanos)

En la actualidad existen diversos sistemas de monitorización continua que ofrecen la posibilidad de medir en continuo valores instantáneos y promediados. El sistema general que presentamos a continuación, aparte de medir niveles sonoros, tiene la funcionalidad añadida de identificar aquellos sonidos que generan una superación de un umbral sonoro configurado (nivel sonoro deseado). Para poder identificar éstos sonidos urbanos que generan una superación de niveles sonoros se ha desplegado un sistema completo que requiere de los siguientes elementos hardware como software:

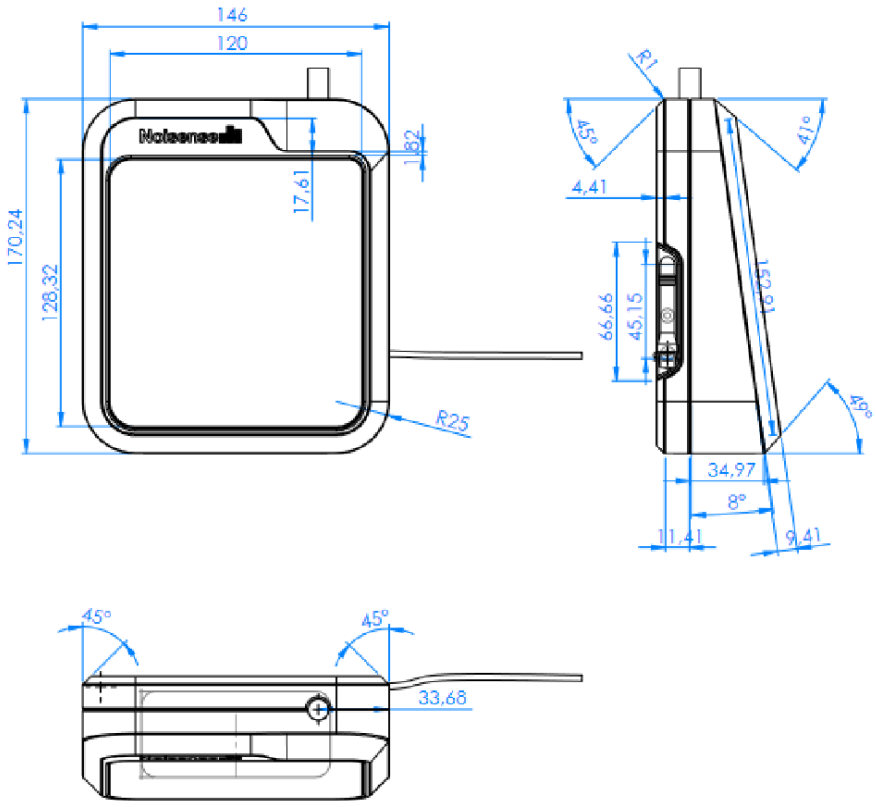
- **Dispositivo NOISENSE** que mide niveles sonoros en continúo y graba audios en cuanto en el sistema se dispara una alarma. Estos audios se guardan en el servidor en la carpeta correspondiente a cada dispositivo instalado.
- **Modelo basado en redes neuronales** ya entrenado y que se encuentra servido en el servidor de NOISMART.
- **Programa cliente,** ubicado en el servidor en el que se llevan a cabo las operaciones necesarias para procesar los datos de la solicitud al modelo y de la respuesta el modelo.



El funcionamiento general del sistema clasificador de sonidos se basa en el uso de la funcionalidad de generación de alarma programada en el dispositivo NOISENSE. Esta alarma se dispara cuando el dispositivo NOISENSE detecta una superación durante 3s consecutivos sobre un umbral pre-configurado. Una vez se dispara la alarma se genera un archivo de audio en formato wav cuya duración depende del fin de la alarma, que se da cuando el dispositivo NOISENSE detecta durante 3 segundos consecutivos unos niveles inferiores al umbral de alarma pre-configurado. El audio se trasmite al servidor.

El dispositivo Noisense, es el sistema que lleva a cabo las medidas de niveles sonoros en continuo, genera alarmas y su audio correspondiente.

El dispositivo tiene las dimensiones que se muestran en la figura y un peso inferior a 2Kg, lo que le hace muy ligero y permite anclar mediante bridas al báculo de alumbrado, de cámara de tráfico, etc., en general a cualquier soporte vertical.



El dispositivo mide los niveles sonoros y graba los audios en formato wav con una frecuencia de muestreo de 48000hz y 16 bits de codificación. Está formado principalmente por una tarjeta PCB comercial Orange Pi PC Plus, que consta de una CPU H3 Quad-core Cortex-A7 H.256/HEVC 4K, 512MB DDR RAM y 8GB EMMC de memoria Flash.

El UMIK miniDSP, es un micrófono comercial de patrón de medición omnidireccional y una cápsula tipo electret de 6mm. Su respuesta frecuencial es de 20-20kHz, resolución de 24 bits y frecuencia de muestreo de 48000Hz, y un nivel máximo SPL para 1% THD a 1khz de 133Dbspl. de sonido máximo en la salida.

2. Sistema Clasificador de Alarmas

El sistema de clasificación de alarmas generadas por sonidos urbanos, es un algoritmo propio desarrollado en lenguaje Python y entrenado con una base de datos generada con el propio dispositivo NOISENSE.

Los audios, tanto de la fase de entrenamiento como los audios que se procesan en producción, se graban con una frecuencia de muestreo de 48kHz y 16 bits de codificación y se almacenan en el servidor en la carpeta destinada al dispositivo.

2.1 Procesamiento de audios

Los audios se procesan para extraer las características acústicas a partir de las cuales el modelo DNN instalado en el servidor identificará el foco sonoro.

- Se segmentan los audios en clips de 1s con un solapamiento de 0.5s.
- Por cada frame se extraen dos tipos de características acústicas: log-mel espectrogramas y openl3 embeddings.
    - Log-mel-espectrograma: por cada frame se calculan transformadas de Fourier en tiempo corto con un enventanado (proceso de selección de un intervalo) de 1024 muestras (21ms) con un solape de 512 muestras (10.6ms), y nos quedamos con el valor de su amplitud en dBs.
    - Una vez se obtiene el espectrograma, se le aplica un filtrado mel de (128filtros). Se aplican filtros mel para tener en cuenta las características perceptuales del oído humano. Se ha utilizado la librería de Python librosa para extraer los parámetros.
    - Openl3 embeddings: estos parámetros se extraen de una red pre-entrenada con sonidos ambientales. El proceso de segmentación es el mismo que para log-mel-espectrograma, pero en lugar de calcular espectrogramas, se calculan los vectores llamados “embeddings”. Estos vectores tienen una longitud de 512.

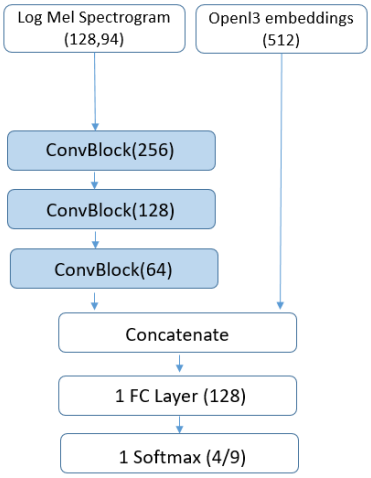
Los datos se normalizan a nivel de frame utilizando la media y la desviación estándar del propio frame. Una vez procesados los audios por cada audio se obtienen dos tensores, uno de 3 dimensiones con los datos referentes a los log-mel-espectrogramas y otro de 2 dimensiones referente a los openl3 embeddings.

3. Arquitectura de los modelos

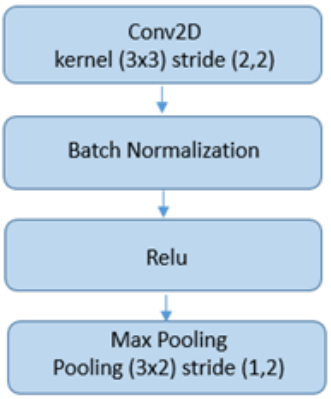
Los dos modelos están implementados en Python 3.8 y tienen un núcleo general que consiste en un modelo de dos entradas. La primera procesa espectrogramas log-Mel, la segunda usa incrustaciones

“embeddings” OpenL3. La diferencia entre los dos modelos servidos es la rama que procesa los log-mel-espectrogramas, uno de los modelos utiliza 3Bloques convolucionales y el otro una ResNet. La salida de la rama que procesa los log-mel-espectrogramas se unen a los “embeddings” para utilizarlos como entrada a la capa oculta de 128 neuronas, y su salida utilizarla como entrada de la capa clasificadora.

- 3Bloques Convolucionales



Los log-mel-espectrogramas se procesan en la rama compuesta por 3 bloques convolucionales con un número de filtros descendente. Cada bloque convolucional del Sistema de detección y clasificación consiste en una capa convolucional 2D con un tamaño de núcleo de (3x3), stride de (2, 2), y padding igual. Después de cada capa convolucional, se utiliza la normalización por lotes, activación ReLu y Max pooling con un tamaño de pool de (3x2) y stride (1, 2). Cada uno de los bloques tiene un número decreciente de filtros convolucionales: 256, 128 y 64.

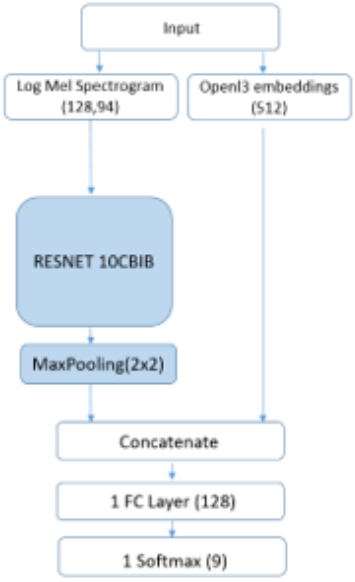


Los openl3 embeddings se concatenan directamente a la salida de los bloques convolucionales. Los datos concatenados se utilizan como entrada a la capa de neuronas (128) completamente conectada, por último se introduce a la capa de salida que tiene el número de neuronas correspondientes a las clases del clasificador (9). Se aplica una función

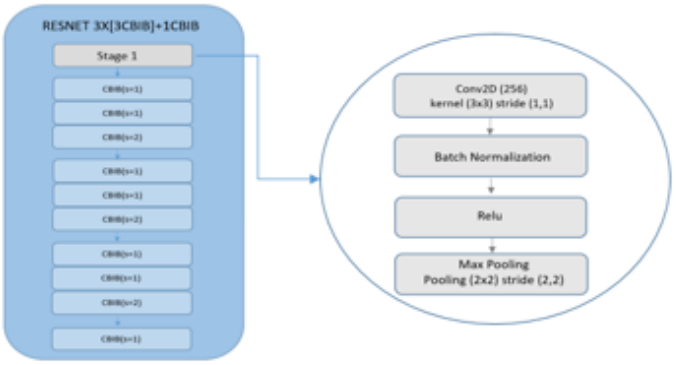
softmax para obtener la probabilidad de cada clase a partir de los scores de las neuronas de salida.

ResNet10CBIB

El ResNet10CBIB consta de una etapa 1, tres bloques convolucionales (CB) y bloques de identificación (IB) con stride (1, 1, 2) repetidos 3 veces más un bloque CBIB de stride 1. Después de todos los bloques, un MaxPooling de 2x2.



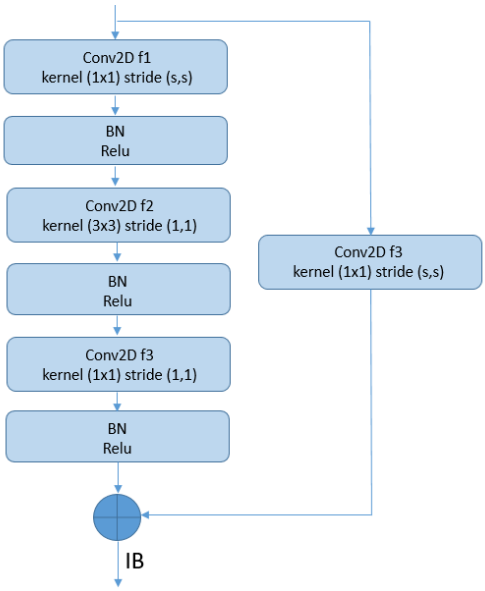
ResNet10CBIB



Stage1 +3CBIB+CBIB

La etapa 1 consta de una capa convolucional 2D (Conv2D) con un número de 256 filtros, un tamaño de kernel de 3x3 y un stride de (1,1). Después de la capa convolucional, se usa la normalización por lotes, la activación de ReLU y una capa MaxPooling de 2x2 y el mismo stride.

Cada bloque convolucional (CB) consta de tres capas convolucionales 2D y una realimentación que consta de una capa convolucional 2D. Los Conv2D de la rama principal tienen un número decreciente de filtros (256,128,64), tamaño de kernel de 1x1 e excepto la segunda capa con un tamaño de kernel de 3x3, stride variable para la primera capa y (1,1) para las demás. La realimentación consiste en un Conv2D con el mismo número de filtros que el tercer tamaño de kernel Conv2D de 1x1 y stride variable. La salida de los bloques CB y el acceso directo se agregan para usarlos como entrada del bloque de identidad.



La configuración del bloque de identidad es la misma que la del bloque convolucional excepto por el atajo. No hay Conv2D para el bloque de identidad.
Al final de la rama de ResNet, se agrega una capa de MaxPooling con un tamaño de kernel de 2x2. La salida del bloque ResNet se concatena con las incrustaciones de OpenL3. El vector de parámetros combinado resultante se utiliza como entrada para la capa de 128 neuronas completamente conectadas. Finalmente, se utiliza una capa softmax (para obtener la probabilidad de cada clase a partir de los scores de las neuronas de salida) para la clasificación.
En los dos casos los modelos tienen una salida de 9 neuronas ya que es capaz de clasificar 9 sonidos urbanos diferentes: cars, motorbikes, cleaning-trucks, voices murmur, music, barking dog, storm, impacts and machinery.

3. Request al modelo

El modelo previamente entrenado se encuentra a la escucha en el servidor de NOISMART, en un contenedor DOCKER. El servidor Cloud consta de 1vCPU, 2GB de RAM y 200GB de disco duro. Sistema operativo ubuntu 18.04. Tiene instalado todo el software necesario para poder servir y ejecutar el modelo.

Se han instalado los siguientes paquetes:

- Argparse
- Csv
- Datetime
- Json
- Gzip
- Os
- numpy as np

- pandas as pd
- keras
- tensorflow as tf
- os
- sys
- sklearn.decomposition
- keras
- resampy
- traceback
- soundfile as sf
- numpy as np
- six
- librosa
- json
- requests
- tensorflow
- tensorflow-serving
- Python 3.8

Una vez se tienen los datos preparados se envía una solicitud de petición al modelo que se encuentra en escucha en el servidor. Para ello se utiliza un objeto json que contiene los dos vectores a procesar por la DNN).

```
r = requests.post('http://localhost:8501/models/test_noisense:predict', json=payload)
```

El modelo responde con otro objeto con los datos de la respuesta decodificado en formato UTF-8. Esa respuesta se procesa para obtener la etiqueta correspondiente a la fuente de sonido que ha generado la alarma. Debido a que el modelo da una predicción por cada frame, mediante majority voting se obtiene la clasificación correspondiente a toda una alarma (audio). Para obtener una única clasificación para toda una alarma se asignan a todos los frames un índice que corresponde al audio al que pertenece. Para cada audio, se cuentan las predicciones que el modelo ha asignado a cada una de las categorías, como resultado final se le asigna la clase correspondiente a la categoría con un mayor número de aciertos.

La salida del modelo es un archivo csv, en el que se muestra el nombre de la alarma y la categoría asignada obtenida de las predicciones del modelo.