Autoencoder convolucional para la depuración de audio

Matías Di Liscia[a]

[a] *Facultad de Ingeniería de la Universidad de Buenos Aires, Buenos Aires, Argentina*

Abstract

Una simple solución utilizando un autoencoder para limpiar señales de audio con abundancia de clicks.

Keywords: autoencoder, red convolucional, audio, tocadiscos

1. Introducción

En esta era contemporánea, es imprescindible el poder adaptar tecnologías analógicas a un mundo en donde predomina lo digital. En particular, se discutirá sobre los discos de vinilo. Un tocadiscos es un dispositivo de reproducción de audio, que mediante el movimiento de una aguja o púa causado por los surcos grabados en el disco de vinilo, genera señales eléctricas que se amplifican y transforman en sonido finalmente. Estas agujas suelen ser de materiales durables, actualmente en general, de diamante.

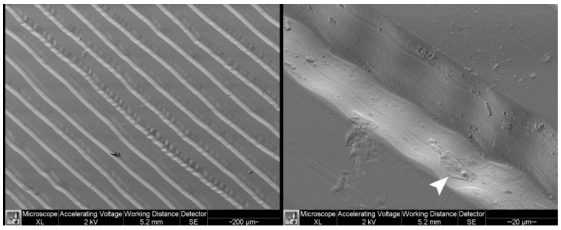

Figura 1: Surcos de disco de vinilo bajo microscopio (6)

Las reiteradas pasadas de ésta sobre los discos a largo plazo generan pequeñas imperfecciones en la pista, imperceptibles al ojo. Pero debido al funcionamiento recién explicado, al estar corriendo el dispositivo éstas moverán la aguja de manera tal que el audio se escuchará distorsionado. Estas distorsiones se perciben como un sonido de "fritura" o clicks, perturbaciones de corta duración y magnitud elevada en comparación con la señal original.

La idea de este artículo es explorar una posible solución a este problema, una vez digitalizado el audio. Mediante el diseño y entrenamiento de un autoencoder convolucional, se pretende demostrar que ésta es una manera simple y fácil de aplicar para resolver el problema, sin tener que procesar el audio a través de complejos algoritmos, confiando en la habilidad de la red neuronal de aprender correctamente la representación del espacio de entrada.

2. Autoencoders

Un autoencoder es una red neuronal utilizada generalmente para comprimir el espacio de entrada en uno de dimensión más pequeña, a partir del cual se puede reconstruir la entrada minimizando la función de pérdida. Su entrenamiento es no supervisado.

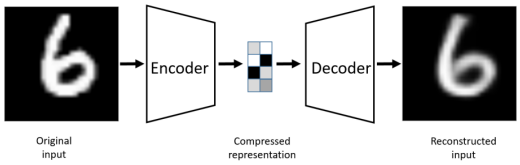

Figura 2: Diagrama de un autoencoder genérico(2)

Consta de tres partes fundamentales: El encoder, el decoder, y la representación comprimida. La dos primeras son redes neuronales, que pueden ser de capa simple o no, para diseñar un modelo profundo y darle robustez. La representación comprimida es una única capa de neuronas (en general de menor tamaño que la primera capa del encoder) que conecta a las otras dos partes. En el caso que se utiliza generalmente, decoder y encoder son perceptrones multicapa, con capas externas del tamaño del espacio de entrada y, de menores tamaño a medida que las capas se acercan al centro.

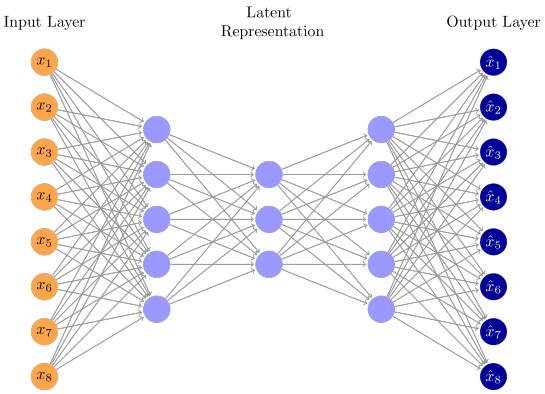

Figura 3: Autoencoder con capas *fully connceted*(7)

27 de febrero de 2024

Durante el entrenamiento, se le muestra a la red un conjunto de elementos representativos del espacio de entrada, y se le "exige" que a la salida haya el mismo elemento que a la entrada. Comparando con la función de pérdida elegida, se hace backpropagation para ajustar los pesos de las neuronas, y el proceso se repite hasta que termine el entrenamiento. Si el sistema es lo suficientemente robusto, podrá encontrar una representación comprimida del espacio de entrada, que se puede reconstruir con la segunda mitad de la red, el decoder.

Existen diversos tipos de autoencoders en la actualidad, cada uno con sus finalidades y relaciones de compromiso propias. Para más detalle, referirse a (2).

2.1. Denoising autoencoders

Un denoising autoencoder es aquel autoencoder cuyo objetivo principal no es el de la compresión, sino el de limpiar los elementos de entrada que vienen contaminados con ruido. Si bien este tipo de red surge como una opción de regularización para evitar overfitting, su finalidad nombrada previamente es ampliamente usada.

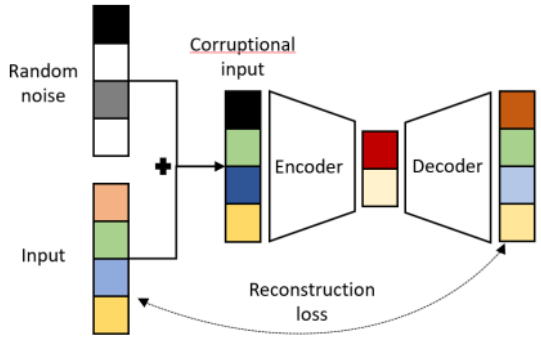

Figura 4: Denoising autoencoder(2)

El principio de funcionamiento es básicamente el mismo que el autoencoder genérico, con la diferencia de que durante el entrenamiento se alimentará a la red con la señal *más* el ruido y a la salida se comparará con la entrada *sin* el ruido, esperando que de esta manera la red entienda que debe suprimirlo. Cabe aclarar que si bien su objetivo es el de la corrección de ruido, la red continúa hallando una representación comprimida de la entrada, aunque para estos tipos de autoencoder suela no usarse. Un problema que se desprende directamente de esto es que la red naturalmente tiende a intentar hallar una buena representación comprimida, y si no se la diseña correctamente, usando un buen set de entrenamiento podría suceder que no depure la entrada, y se preocupe más por comprimir. Esto se discutirá más adelante, y se verá potenciado por el hecho de que los clilcks analizados como ruido tienen poca energía respecto a la señal de audio limpia.

Respecto a los detalles matemáticos y estadísticos del funcionamiento del denoising autoencoder consultar (5)

2.2. Autoencoder convolucional

Un autoencoder convolucional es cualquier autoencoder cuyo decoder y encoder son redes neuronales convolucionales (CNN). Debido a que el funcionamiento del autoencoder en sí es exactamente el mismo, no se pretende entrar mucho en detalle sobre cómo funciona una CNN (8)(3). Sin embargo, a grandes rasgos un red convolucional funciona de la siguiente manera. La entrada pasa por una primera etapa, en la cual un filtro o *kernel* se utilizará para realizar la convolución (de dimensión correspondiente) sobre ella. A la señal resulante se le suelen tomar fragmentos y realizar *pooling*, para reducir la dimensionalidad aún más. Finalmente, esta última señal será la entrada a la próxima capa. Para la sección del decoder se debe hacer el proceso inverso, espejado. Se define entonces la deconvolución o convolución transpuesta (9), que dado el kernel y una señal genera otra de mayor dimensión.

3. Solución: Denoising Autoencoder Convolucional

Para el problema detallado en la sección 1, se diseñó un autoencoder convolucional que pueda resolverlo. Si bien este tipo de autoencoder se suele utilizar generalmente cuando se trabaja con imágenes (con mucha efectividad), demostró un buen rendimiento para este caso.

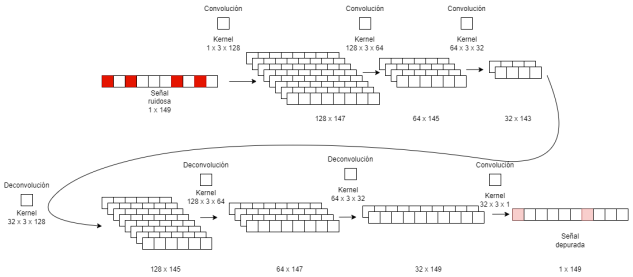

Figura 5: Denoising autoencoder convolucional utilizado

El modelo consiste de tres capas de convolución en el encoder, con 128, 64 y 32 filtros de tamaño 3. Dado al tamaño del filtro, la dimensión de la señal se reduce en solamente 2 muestras por capa. Aunque pueda resultar poco, hay que tener en cuenta que las anomalías a remover son de duración muy corta, de aproximadamente 50 muestras[1], por lo que un filtro muy grande podría resultar en que la red pierda información importante sobre ellos en la convolución. Entre estas capas, no se agregó pooling, y los filtros se inicializaron aleatoriamente con una distribución uniforme He. La activación de las neuronas es mediante una *leaky* ReLU. El decoder, también tiene 128, 64, 32 filtros en sus tres capas realizando la deconvolución, y se le agrega al final una capa de convolución activada por la función sigmoidea $\sigma(x) = \frac{1}{1+e^{-x}}$, para pasar de 32 señales a una única señal, que será la salida depurada del sistema. Para todas las capas, se limita el valor absoluto de los pesos a 10, para evitar overfitting.

3.1. Preparación de datos

Para que el entrenamiento de la red sea adecuado, es imprescindible contar un set de entrenamiento óptimo. Para este traba-

[1] Muestreando a 44100Hz.

jo, se tomaron fragmentos canciones de música clásica. Muestreando a 44100Hz se los separó en señales de 149 muestras. Para simular el ruido de clicks, se muestreó un audio real proveniente de un tocadiscos y se le separaron solamente los clicks[2]. A las señales limpias se le sumaron fragmentos aleatorios de la señal exclusiva de ruido. Para ayudar a la red con la identificación del ruido se tuvo en cuenta:

1. El largo de las señales. Se lo eligió pequeño (149 muestras representa aproximadamente 3,4ms) para que la red no tenga problema en distinguir las anomalías, de 30-50 muestras cada una.
2. La energía de los clicks. Con el mismo fin, se eligió una SNR señal-clicks de 20dB, haciendo que los clicks se escuchen más fuertes de lo que se escuchan en la realidad.
3. La energía del ruido blanco. Además, a la señal limpia se le agrega ruido blanco gaussiano, ya que casi cualquier audio real digitalizado de un tocadiscos inherentemente viene contaminado de esta forma. La SNR[3] señal-ruido blanco elegida es más alta, de 60dB.

Además de esto, los datos pasaron por un proceso de normalización. Los audios digitalizados están en formato Waveform Audio Format (o `.wav`), en donde cada muestra es un entero signado de 16 bits. El autoencoder diseñado solamente puede recibir datos entre 0 y 1, además de generar a la salida señales en ese mismo rango por la activación utilizada. Es por esto que antes de que los datos pasen por la red hubo que encontrar una forma de que cumplan con las condiciones necesarias.

Se recorren todas las señales (limpias y ruidosas) y se buscan los valores máximos y mínimos M y m. Finalmente se realiza la transformación:

$$s_{norm}(t) = \frac{s(t) - m}{M - m} \tag{1}$$

En donde el valor 1 lo tomará la máxima muestra de todas las señales y el 0 la mínima. Todos los valores intermedios se transforman linealmente en ese intervalo.

Como ya se mencionó antes, existen problemas a la hora de diseñar un autoencoder que depure. Si bien una pérdida (error cuadrático medio en este caso) del orden de 10^{-4}, que es la que se obtenía en diseños previos de la red, puede ser aceptable para otra aplicación, se vió que para la limpieza de clicks no fue suficiente. Los clicks son anomalías de muy corta duración, y sumado a que el proceso de normalización previo comprime a las señales pérdidas de ese orden pueden ser engañosas. El problema residía en el hecho de que dado a que las señales de entrenmiento eran muy extensas en el tiempo (y los clicks muy cortos y con poca energía) la red no lograba identificarlos correctamente y por lo tanto no los eliminaba, sino que solamente hacía el trabajo de encontrar una representación comprimida, que no es de interés en este caso. Es por esto que se utilizaron señales de 149 muestras, con clicks un poco exagerados al oído, para asegurarse que la red los identifique.

4. Resultados

Finalmente, se entrena a la red con 40000 señales limpias, y sus respectivas versiones ruidosas, y 4000 señales de testeo. Se entrena por 200 épocas con un *batch size* de 4, obteniéndose así una pérdida:

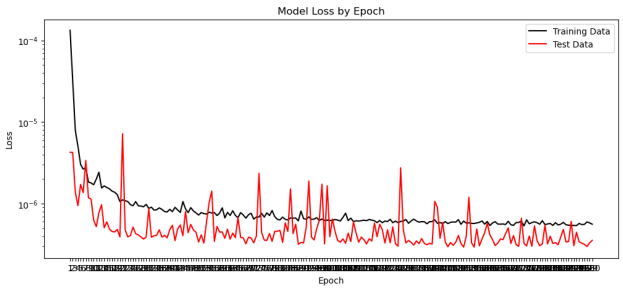

Figura 6: Error cuadrático medio en cada época

Con pérdida rondando los 10^{-7}, ya se obtienen buenos resultados. Por ejemplo tomando señales del set de testeo individualmente:

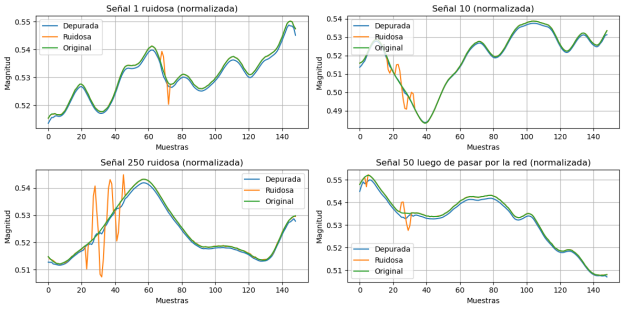

Figura 7: Comparación de set de testeo

O también juntando un fragmento entero:

[2]El problema de identificación y separación de clicks no es trivial, y el algoritmo usado no está optimizado para removerlos, pero sí da una buena idea de cómo son los cilcks en el dominio temporal.

[3]Ambas relaciones señal a ruido se eligieron de forma que el funcionamiento de la red sea el mejor posible. Sin embargo, son parámetros que se pueden modificar fácilmente.

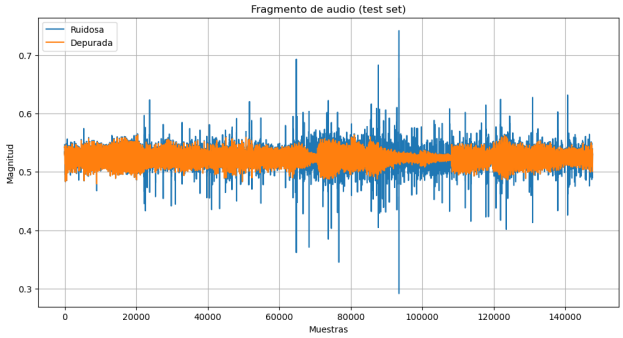

Figura 8: Comparación de fragmento completo de testeo

Para corroborar que está solución se desempeña correctamente, se utilizaron audio reales con clicks para ver si ésta los podía eliminar.

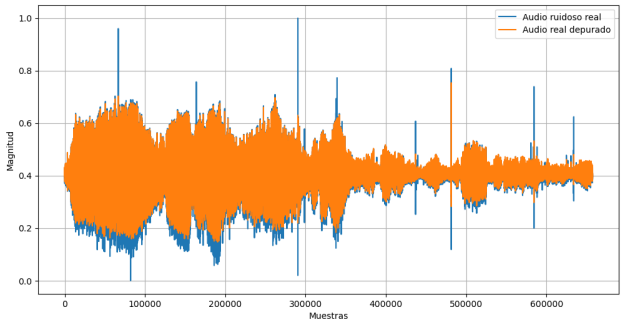

Figura 9: Audio real con clicks (normalizado) - Antes y después

Se ve que los clicks se eliminan bien, a excepción de algunos muy pronunciados sobre el final, alrededor de las 480000 y 580000 muestras. Después de experimentar un poco, se descubrió que la red tiene problemas particularmente para eliminar los clicks más pronunciados en zonas donde la señal útil tiene poca energía. Esto puede ser un problema, ya que es común que las canciones tengan secciones con silencio, o con volúmenes reducidos. Sin embargo, en la sección 5 se proponen algunas alternativas.

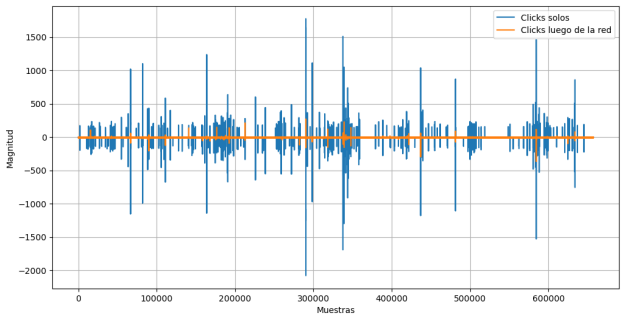

Figura 10: Clicks sobre silencio (normalizados) - Antes y después

En la figura anterior, se aprecia claramente como a la red le cuesta trabajo identificar los clicks cuando éstos son de magnitud elevada y no hay señal útil. Este comportamiento es esperable conociendo la manera en la que se entrenó la red. Todas las señales del set de entrenamiento tenían una SNR fija. En el caso de clicks sobre silencio, la SNR sería muy pobre, por lo que en el set de entrenamiento no había casos en donde esto se cumpla.

Comparando las señales en el dominio de la frecuencia:

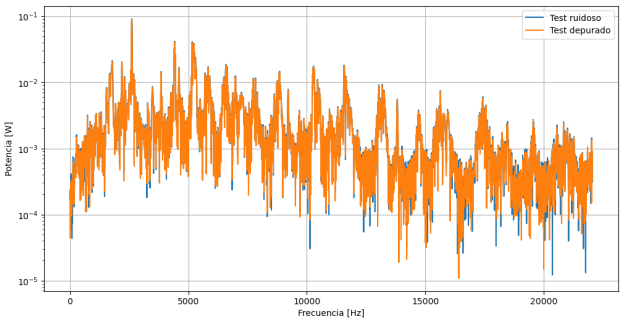

Figura 11: Espectro de audio de testeo

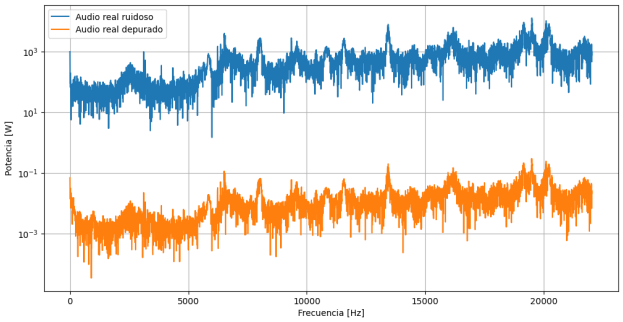

Figura 12: Espectro de audio real

Se puede ver que la forma de los espectros de las figuras 11 y 12 se mantiene en gran medida igual, sobre todo en bajas frecuencias, siendo que es esperable ver mayores discrepancias en altas, dado que los clicks son un fenómeno impulsivo. El que los espectros sean similares remarca el hecho que la red no distorsiona la señal significativamente, aunque sí elimina los clicks, manteniéndose así la integridad de la misma.

La discrepancia entre el valor medio de la potencia en la figura 12 se debe al algoritmo que usa Python para transformar las señales normalizadas en archivos `.wav`. La señal con ruido nunca pasó por ese proceso, por lo que tiene mayor energía media.

Por último, se deja una tabla comparando la cantidad de clicks antes y después de pasar por la red. Como ya se mencionó, definir qué es un click no es simple, pero para los fines de confeccionar la tabla cualitativamente, *un click es un conjunto de 50 muestras que contiene al menos una instancia en la cual se cumple*:

$$\frac{|s[n+1]|}{|s[n+1]| + |s[n]|} > u \tag{2}$$

En donde u es un umbral de la sensibilidad definido por el usuario, y s la señal de interés. Esta definición es buena, pero hay que ajustar a mano el umbral, por lo que solamente es útil a los fines de confeccionar la tabla:

Fuente	clicks/muestra antes	clicks/muestra después
Test set	0,0102	0
Audio real	$1{,}3687 \cdot 10^{-5}$	$3{,}046 \cdot 10^{-6}$

Cuadro 1: Comparación de cantidad de clicks

5. Conclusiones y consideraciones para futuros trabajos

El presente trabajo logró mostrar una solución simple a un problema muy complejo de resolver si no se hace uso de la capacidad de aprendizaje de las redes neuronales. Se vio en evidencia la importancia de caracterizar el problema correctamente antes que nada, y luego poder generar un conjunto de datos que se ajuste a las conclusiones sacadas.

A pesar de todo esto, se mostraron también algunas limitaciones de la red, por lo que si el lector quiere continuar con el trabajo de depuración se hacen algunas sugerencias.

El gran problema era el ruido en zonas con poca señal útil. Una opción para resolverlo podría ser el de entrenar otro autoencoder de características similares, pero entrenarlo con fragmentos de audio con SNR pobre, o directamente con audios sin señal útil. Una vez entrenada, usarla para volver a pasar los audios depurados (solamente las secciones problemáticas) por la red de este informe. Si bien se puede entrenar una única red tanto con el set usado para este trabajo como con el sugerido recientemente al mismo tiempo, no es recomendable, ya que afectara la capacidad de la misma de identificar los clicks.

Otra solución sería hacerle un preprocesamiento a la señal. Mirando las ideas de (4), un filtrado lineal puede ayudar a disminuir un poco la magnitud de los impulsos, ayudando posteriormente a la red a eliminarlos. Otra opción un poco más sofisticada también explicada en la referencia anterior es la identificar los clicks con un algoritmo similar al utilizado para confeccionar la tabla 1, eliminarlos y reemplaazar ese fragmento con otro que sea cercano y tenga alta correlación con el eliminado. Esto de por sí tiene sus problemas también, ya que es difícil ajustar el umbral de decisión para encontrar *todos* los clicks, y que además en el reemplazo podría pasar que el fragmento con más correlación sea otro click. Sin embargo, un preprocesamiento con esta técnica ayudaría ampliamente a la red eliminando los clicks más violentos.

Para finalizar, se considera que el presente trabajo logra resolver el problema planteado satisfactoriamente, dando también opciones para expandirlo y mejorarlo en el futuro.

Apéndice A. Comentarios sobre el código

El código con el que se diseñó el autoencoder está en el lenguaje Python, utilizando la librería TensorFlow. Si bien éste está comentado y es fácil de entender, se procede a explicar brevemente la utilización de las funciones del archivo `datos_con_clicks.py`, creado específicamente para facilitar el confeccionado de este trabajo.

El archivo contiene funciones auxiliares que en su mayoría se relacionan con el preparado de los datos con los que se alimenta a la red.

- `normalizar(train, train_noisy, test, test_noisy)`: Recibe cuatro listas de listas, y las normaliza de acuerdo a la ecuación 1. Devuelve cuatro arrays de Numpy equivalentes a la normalización de los parámetros recibidos, y el máximo y mínimo hallado, en caso de que se quiera desnormalizar. Internamente la función primero escala los datos, para evitar un overflow eventualmente debido al formato `.wav` de donde se originaron los datos.

- `add_noise(datos, ruido, snrb, snrc, energia_datos)`: Dado una señal útil (`datos`) y su energía le agrega ruido blanco de acuerdo a la SNR recibida como `snrb` y clicks de acuerdo a la otra SNR recibida `snrc` a partir de la señal de clicks `ruido`. Debido a la aleatoreidad a la hora de elegir un click y a cómo está programada la función, existe una probabilidad muy pequeña (del 0.02 aproximadamente) de que no se sume ningún click. A pesar de que podría parecer un bug en principio, se dejó así ya que el hecho de que no todas las señales de entrenamiento tengan clicks (pero **sí** la gran mayoría) ayuda a que los audios artificialmente generados suenen más reales. Si esto no fuera así, la contribución de los clicks suena más parecida a estática o ruido blanco.

- `load_data(n, snrb, snrc, t)`: Genera dos listas de `n` listas en donde cada una de estas últimas representa un fragmento de `t` segundos muestreado a $44100Hz$, de audios sin ruido y con ruido de acuerdo a las SNR recibidas. Se asume que los archivos de audio son `.wav`, sus nombres son solamente números y se encuentran en una carpeta llamada `data_limpia`. También se asume que en la misma dirección que este archivo de funciones se encuentra el audio que contiene los clicks solos `ruido.wav`.

- `deteccion(s, sensibilidad)`: Devuelve una lista con la ubicación de los clicks dentro de la señal `s` de acuerdo al parámetro de la sensibilidad, conforme a la ecuación 2.

Referencias

[1] Travis Boylls. How record players work: Turntables, vinyl pressing, & more. `https://www.wikihow.com/How-Do-Record-Players-Work#How-Record-Players-Work`, 2023. Understand how vinyl records make sound when played on turntables.

[2] Raja Giryes Dor Bank, Noam Koenigstein. Autoencoders. 2021.

[3] Ryan Nash Keiron O'Shea. An introduction to convolutional neural networks. 2015.

[4] Matías Di Liscia. Trabajo práctico especial - Señales y Sistemas. 2021.
[5] Yoshua Bengio Pierre-Antoine Manzagol Pascal Vincent, Hugo Larochelle. Extracting and composing robust features with denoising autoencoders. 2008.
[6] Furnace Record Pressing. Vinyl 101: How to prepare your audio for vinyl. https://www.furnacemfg.com/vinyl-record-audio-preparation/, 2024.
[7] Janosh Riebesell. Autoencoder. https://tikz.net/autoencoder/, 2021.
[8] Sumit Saha. A comprehensive guide to convolutional neural networks — the ELI5 way. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53, 2018.
[9] Francesco Visin Vincent Dumoulin. A guide to convolution arithmetic for deep learning. Sección 4. 2016.