Aplicaciones de la Biometrial de Voz

Identificación de pacientes de párkinson a través de la voz mediante redes neuronales

UNIVERSIDAD
POLITÉCNICA
DE MADRID

Julia Serrano Arrese c200119
Jose Alberto Cañibano Lopez a180192

Índice

1. Introducción

La investigación que vamos a realizar se basa en el estudio de un conjunto de audios de la voz de pacientes con Parkinson. El objetivo será determinar si, por medio de espectrogramas y gráficas, se pude identificar la enfermedad frente a una muestra de control.

En total hemos podido trabajar con 73 archivos de audios. De los cuales 42 son de la muestra de control y 31 son de pacientes con Parkinson. Es importante mencionar que, aunque la cantidad de muestras de pacientes sea pequeña la longitud de cada una de estas es extensa, lo que nos ha permitido un mejor ajuste de los datos.

Para la realización del estudio hemos usado el entorno de programación Anaconda, donde por medio de Jupyter Notebooks hemos creado un entorno virtual de TensorFlow, el cual nos ha facilitado mucho la realización.

2. Tratamiento de los datos

El conjunto de datos que tenemos pertenece al dataset MDVR-KCL (Mobile Device Voice Recordings at King's College London).

Para usarlos los descargaremos al ordenador y comenzaremos dándoles formato. En primer lugar, dividiremos los datos que tenemos en HC, Control, y PD, Pacientes enfermos. En segundo lugar y para poder procesarlos deberemos convertirlos de audio a 16 bits. Teniendo en cuenta que:

- Cada archivo WAV contiene datos de series temporales con un número determinado de muestras por segundo
- Cada muestra representa la amplitud de la señal de audio en ese momento específico
- En un sistema de 16 bits, como los archivos WAV en el conjunto de datos mini Speech Commands, los valores de amplitud oscilan entre -32 768 y 32 767.
- La frecuencia de muestreo para este conjunto de datos es de 16 kHz.

Debido a la longitud de los archivos hemos optado por crear una función para poder dividir y seleccionar aleatoriamente secciones de los audios que posee el dataset. Además dividiremos también los archivos entre sets de entrenamiento, validación y prueba.

3. Visualización de ondas de audio

Para realizar esto haremos uso de una función auxiliar que tiene de entrada un fragmento de audio y a salida es una tupla que contiene el audio y los tensores de etiquetas listos para el aprendizaje supervisado. Que junto con las funciones *get_waveform* y *get_waveform_from_fragment* obtendremos la waveform a partir de un tensor dado y de un fragmento dado respectivamente.

Una primera visualización de una onda de audio a partir de un fragmento de entrenamiento aleatorio seria la siguiente:

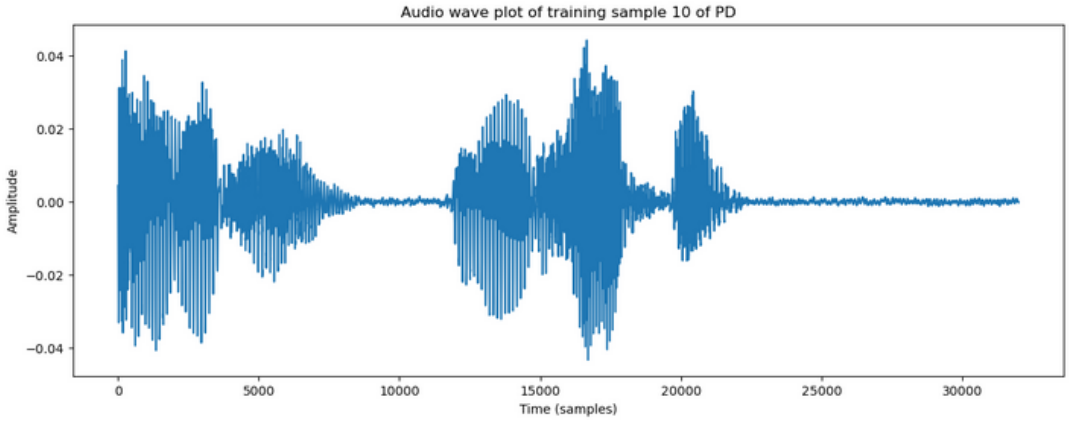

Figura 1: Waveform fragmento aleatorio

Y en esta segunda figura visualizamos las 9 ondas de audio aleatorias del train set:

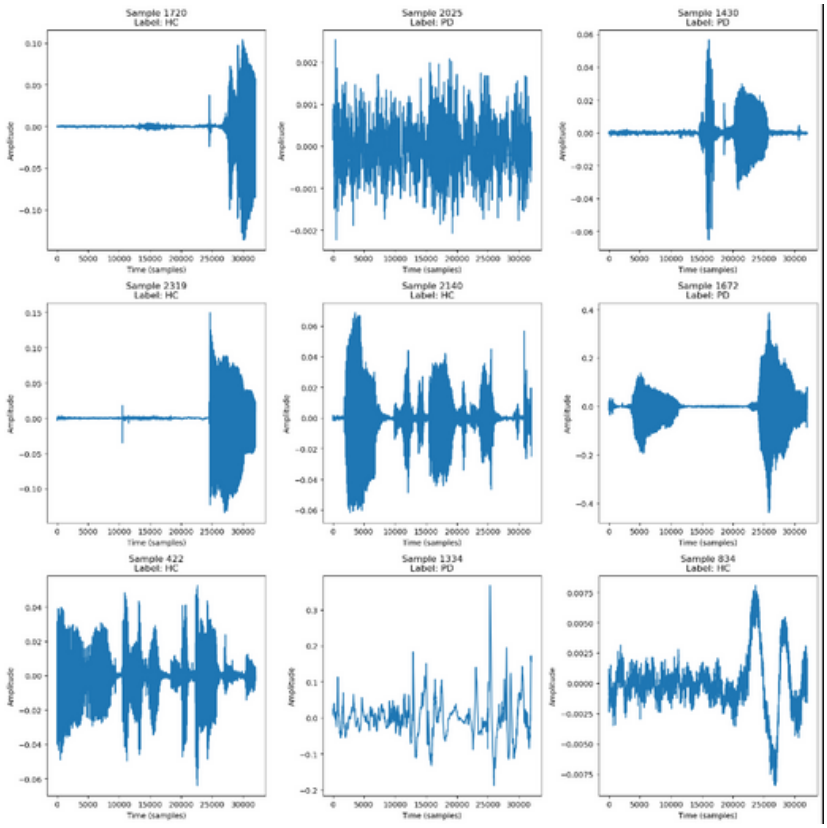

Figura 2: Waveforms del Train set

4. Visualización de Espectrogramas

Una vez obtenidos los “waveform” debemos obtener los espectrogramas. Se realizará mediante la aplicación de STFT (short-time Fourier transform). Para realizar esta aplicación programaremos un único método que nos realice esta transformación de Fourier. Lo primero que debemos hacer es comprobar que ambas “waveform” son del mismo tamaño. Esta función nos deberá devolver un array de números complejos que representan la magnitud y la fase, pero nosotros solo debemos utilizar la magnitud. Una vez ya tenemos hecho todo esto podemos escuchar un audio y sacar tanto su “waveform” y su espectrograma.

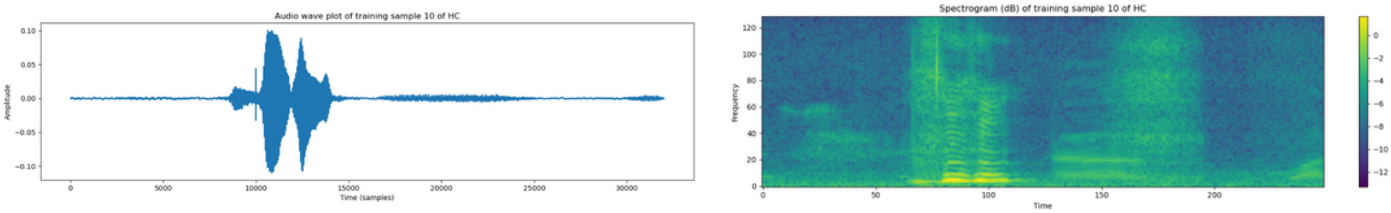

Figura 3: Comparación entre Waveform y Espectrograma

A continuación podemos ver 9 espectrogramas con sus correspondientes etiquetas:

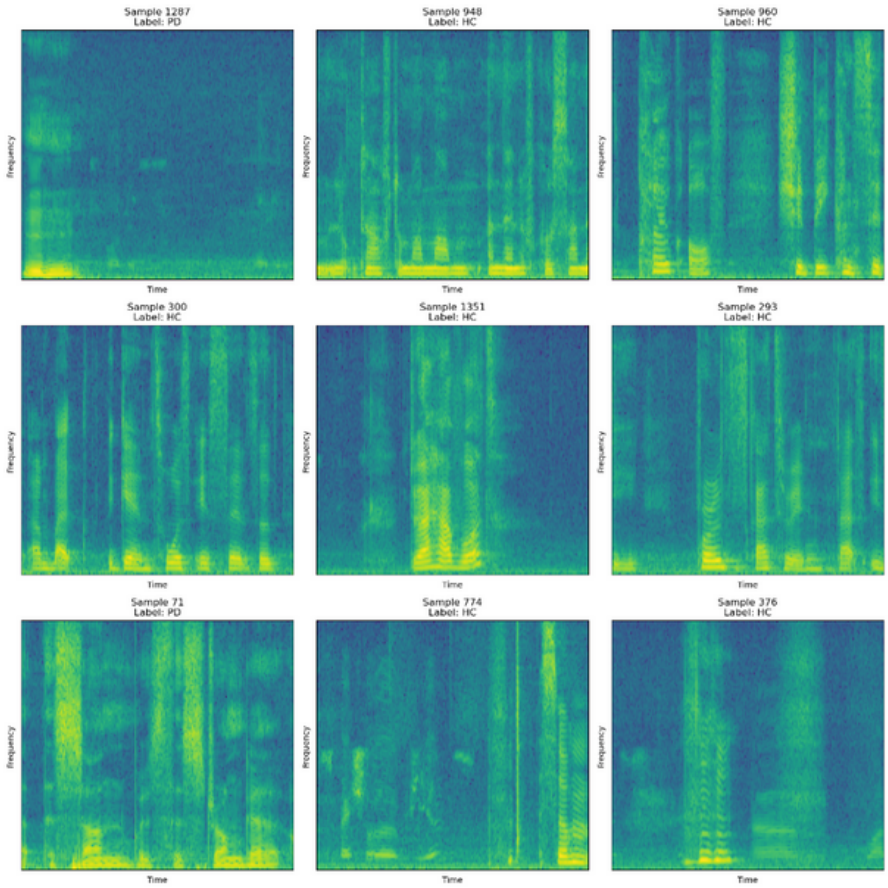

Figura 4: Espectrogramas del Train set

5. Preprocesamiento de datos

Para el modelo, utilizamos una red convolucional (CNN) simple, ya que hemos transformado los audios .wav a espectrogramas.

El modelo *tf.keras.Sequential* utiliza las siguientes capas de preprocesamiento:

- *tf.keras.layers.Resizing:* para reducir la muestra de la entrada y permitir que el modelo se entrene más rápido.
- *tf.keras.layers.Normalization:* para normalizar cada píxel de la imagen en función de su media y desviación estándar.

Para la capa de *Normalization*, primero llamamos al método *adapt* en los datos de entrenamiento para calcular las estadísticas agregadas (media y desviación estándar).

```
Input shape: (64, 129, 249)
Model: "sequential_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resizing_10 (Resizing)      (None, 32, 32, 1)         0

 normalization_12 (Normaliza  (None, 32, 32, 249)      499
 tion)

 conv2d_20 (Conv2D)          (None, 30, 30, 32)        71744

 conv2d_21 (Conv2D)          (None, 28, 28, 64)        18496

 max_pooling2d_10 (MaxPoolin  (None, 14, 14, 64)       0
 g2D)

 dropout_20 (Dropout)        (None, 14, 14, 64)        0

 flatten_10 (Flatten)        (None, 12544)             0

 dense_20 (Dense)            (None, 128)               1605760

 dropout_21 (Dropout)        (None, 128)               0

...
Total params: 1,696,757
Trainable params: 1,696,258
Non-trainable params: 499
```

Figura 5: Capas del modelo

6. Entrenamiento del modelo

Para el entrenamiento del modelo, utilizamos la biblioteca Tensorflow y la API de Keras. Se utilizó el optimizador Adam y la pérdida de entropía cruzada para configurar el modelo.

Entrenamos al modelo durante 10 épocas y utilizamos el conjunto de datos de entrenamiento y validación para medir la precisión y la pérdida.

```
Epoch 1/10
37/37 [==============================] - 7s 173ms/step - loss: 1.5762 - accuracy: 0.4884 - val_loss: 0.6942 - val_accuracy: 0.4932
Epoch 2/10
37/37 [==============================] - 6s 172ms/step - loss: 0.7014 - accuracy: 0.5411 - val_loss: 0.6805 - val_accuracy: 0.5925
Epoch 3/10
37/37 [==============================] - 6s 167ms/step - loss: 0.6846 - accuracy: 0.5732 - val_loss: 0.6688 - val_accuracy: 0.5993
Epoch 4/10
37/37 [==============================] - 6s 166ms/step - loss: 0.6966 - accuracy: 0.5668 - val_loss: 0.6461 - val_accuracy: 0.5959
Epoch 5/10
37/37 [==============================] - 6s 166ms/step - loss: 0.6675 - accuracy: 0.5920 - val_loss: 0.6459 - val_accuracy: 0.6130
Epoch 6/10
37/37 [==============================] - 6s 167ms/step - loss: 0.6471 - accuracy: 0.6173 - val_loss: 0.6323 - val_accuracy: 0.6473
Epoch 7/10
37/37 [==============================] - 6s 167ms/step - loss: 0.6033 - accuracy: 0.6759 - val_loss: 0.5926 - val_accuracy: 0.6781
Epoch 8/10
37/37 [==============================] - 6s 170ms/step - loss: 0.5472 - accuracy: 0.7230 - val_loss: 0.5543 - val_accuracy: 0.7158
Epoch 9/10
37/37 [==============================] - 6s 169ms/step - loss: 0.5347 - accuracy: 0.7372 - val_loss: 0.5390 - val_accuracy: 0.7226
Epoch 10/10
37/37 [==============================] - 6s 167ms/step - loss: 0.4943 - accuracy: 0.7688 - val_loss: 0.5309 - val_accuracy: 0.7397
```

Figura 6: Modelo entrenado durante 10 épocas

Para verificar la mejora del modelo durante el entrenamiento, se trazaron curvas de pérdida de entrenamiento y validación. A través del siguiente gráfico podemos comprobar que el modelo está generalizando bien y no está sobreajustado a los datos de entrenamiento, ya que la línea de la pérdida de validación se mantiene cerca a la línea de la pérdida de entrenamiento. Por otra parte, también queda comprobado que el modelo mejora con cada época, ya que las líneas se continúan reduciendo

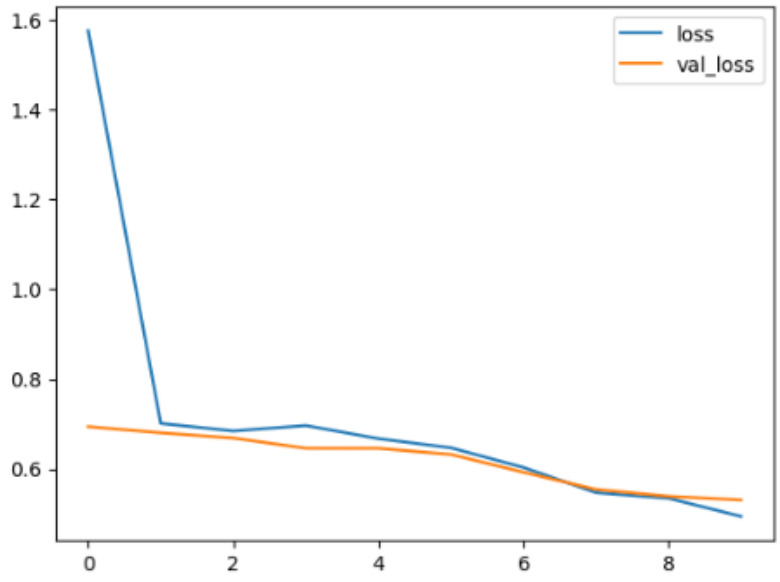

Figura 7: Gráfico de la función de pérdida

7. Evaluación del rendimiento

Ejecutamos el modelo en el conjunto de prueba y obtuvimos una precisión del 71%. Además, utilizamos una matriz de confusión para evaluar el desempeño del modelo. La diagonal principal nos muestra los audios que han sido predichos correctamente (132 HC y 74 PD), y en la diagonal secundaria se muestran por el contrario aquellos audios que el modelo ha predicho incorrectamente (50 HC y 36 PD). Para visualizar esta matriz de confusión utilizamos la biblioteca Seaborn.

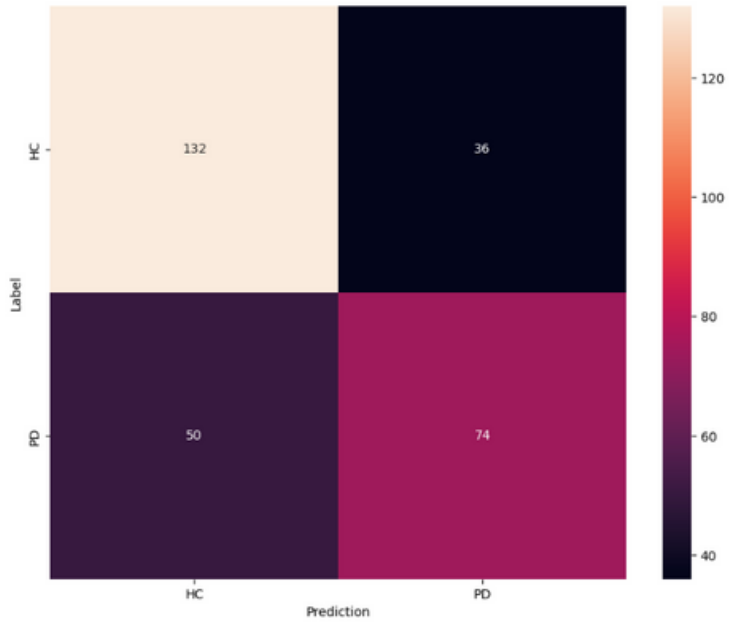

Figura 8: Matriz de confusión

A continuación, ejecutamos inferencia en un archivo de audio para comprobar cómo el modelo realiza su predicción. Trazamos un gráfico de barras para mostrar la probabilidad de cada clase.

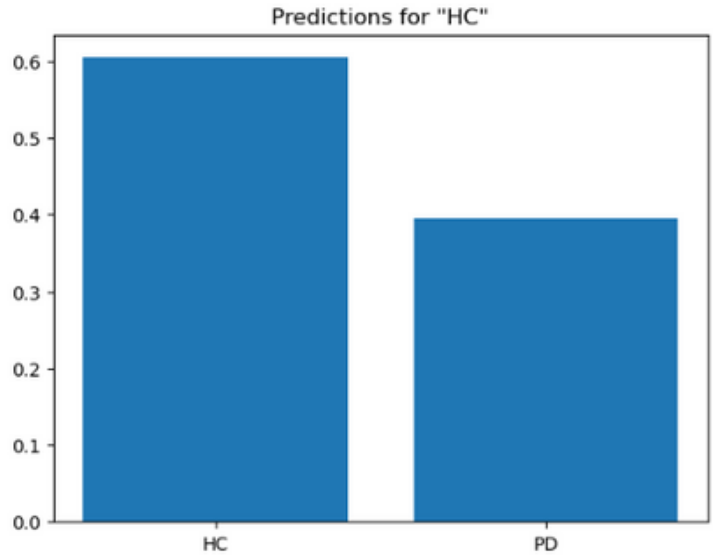

Figura 9: Predicción del modelo

8. Conclusiones

Basándonos en los resultados obtenidos, podemos concluir que nuestro modelo de red neuronal es capaz de identificar pacientes de Parkinson a partir de su voz con una precisión del 71%. Este resultado sugiere que existe una relación entre la enfermedad de Parkinson y las características de la voz.

Sin embargo, todavía hay margen de mejora para nuestro modelo, como la incorporación de más datos de pacientes o la incorporación de información clínica adicional de los pacientes, como la edad o el género, ya que la enfermedad de Parkinsosn puede afectar a cada paciente de manera diferente.

En general, el enfoque de utilizar técnicas de aprendizaje automático para la detección de enfermedades a través de la voz, pensamos que puede tener un gran potencial y podría aplicarse a otras enfermedades.