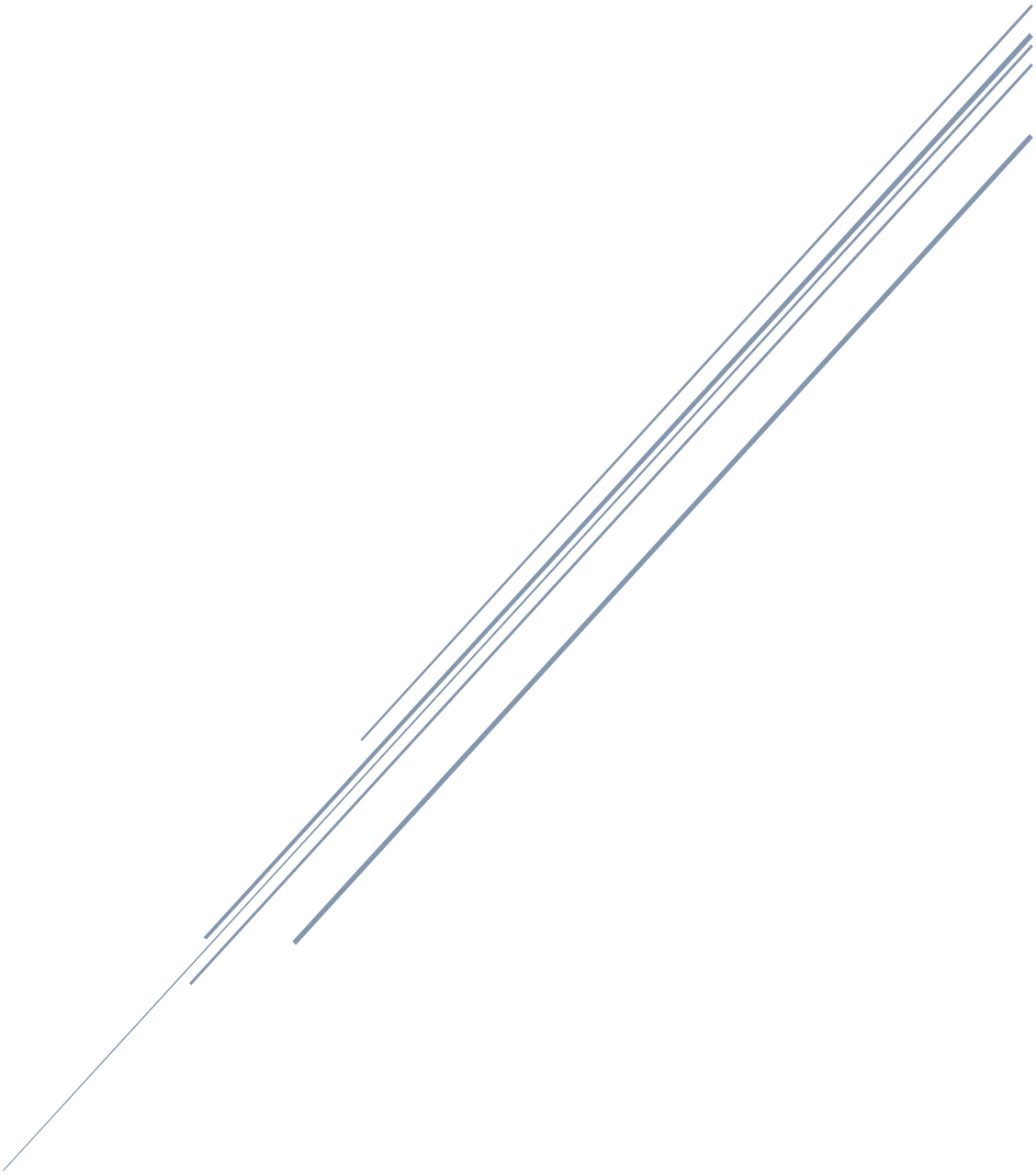
APLICACIONES DE LA BIOMETRÍA DE LA VOZ

Detección de Locuciones Simuladas Mediante Redes Neuronales



Autores:
Adrián Rivero Moreno
José Antonio Márquez Ordóñez
Kevin Alberto López Porcheron

Índice

1. Problema por tratar

Nuestro problema por tratar es la detección de locuciones simuladas mediante redes neuronales que consiste en la realización de experimentos de detección de suplantación de locutores por medio de locuciones simuladas generadas a través de diferentes técnicas.

Para ello, vamos a trabajar con *TensorFlow-Keras*, donde el entorno de trabajo será "*Google Colab*". El motivo es que, este entorno, nos permitirá y nos facilitará la construcción y la ejecución de código por bloques, aunque tendremos una RAM y disco limitados, así que habrá operaciones de grandes magnitudes que no se podrán hacer y habrá que buscar alternativas.

2. Datos empleados

Los datos que vamos a emplear, en la resolución del problema, son un repertorio de audios en formatos flac que están almacenados en "*UPMdrive*".

Para trabajar con ellos los descargaremos para tenerlos de forma local. Tras la descargar de los datos, se realiza una división de los datos, haciendo que tengamos 40.000 casos de entrenamiento para un modelo y 35.526 casos de entrenamiento para un segundo modelo. Esta división se justifica por la limitación del entorno de ejecución (descritas en el anterior apartado), por lo que no podemos entrenar un modelo con todos los casos de entrenamiento posibles. Por este motivo, utilizaremos dos modelos con, especificaciones idénticas, para entrenar los distintos grupos de casos. Finalmente compararemos los resultados para saber si tenemos una red neuronal que funciona correctamente. También extraemos 4.000 casos para validación y 8.000 casos para realizar los test.

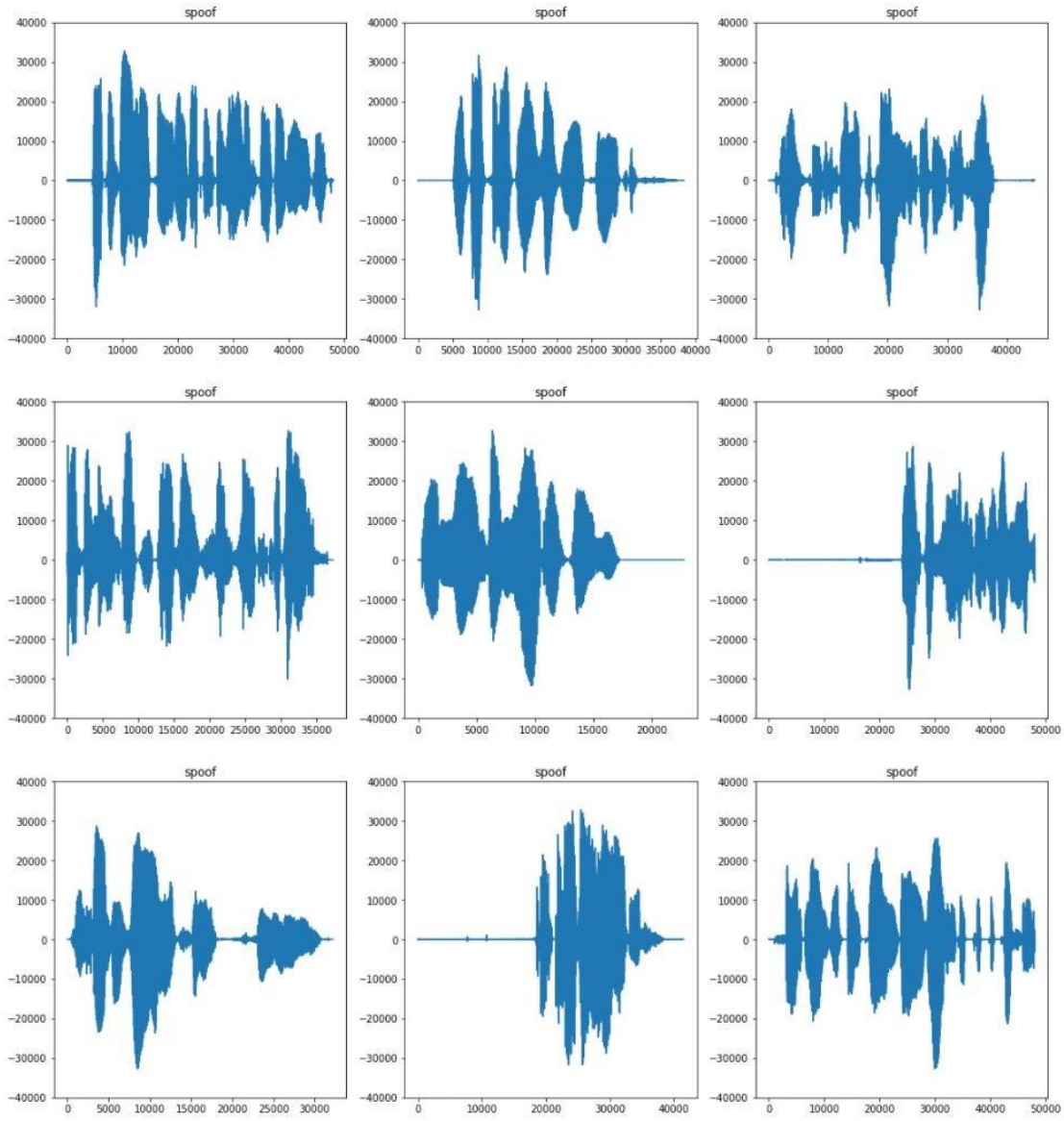
Por último, hay que destacar que hemos definido dos etiquetas: spoof y bonafide; que estarán asociados a cada uno de los audios para poder entrenarlos y poder comprobar si los resultados son correctos.

En cada caso se han sacado únicamente las primeras 48.000 muestras que, con una frecuencia de muestreo de 16.000 muestras por segundo, serian 3 segundos por cada audio. En caso de que un audio no tenga 48.000 muestras se rellenará con ceros.

3. Rasgos que se extraen en el proceso

En este apartado se va a mostrar los rasgos que se extraen de cada audio. Para que la red neuronal pueda trabajar con los datos, necesitamos sacar de cada audio su correspondiente espectrograma con su etiqueta asociada.

Para poder sacar los espectrogramas tendremos que obtener en primer lugar los "*waveform*" asociados a los correspondientes audios:

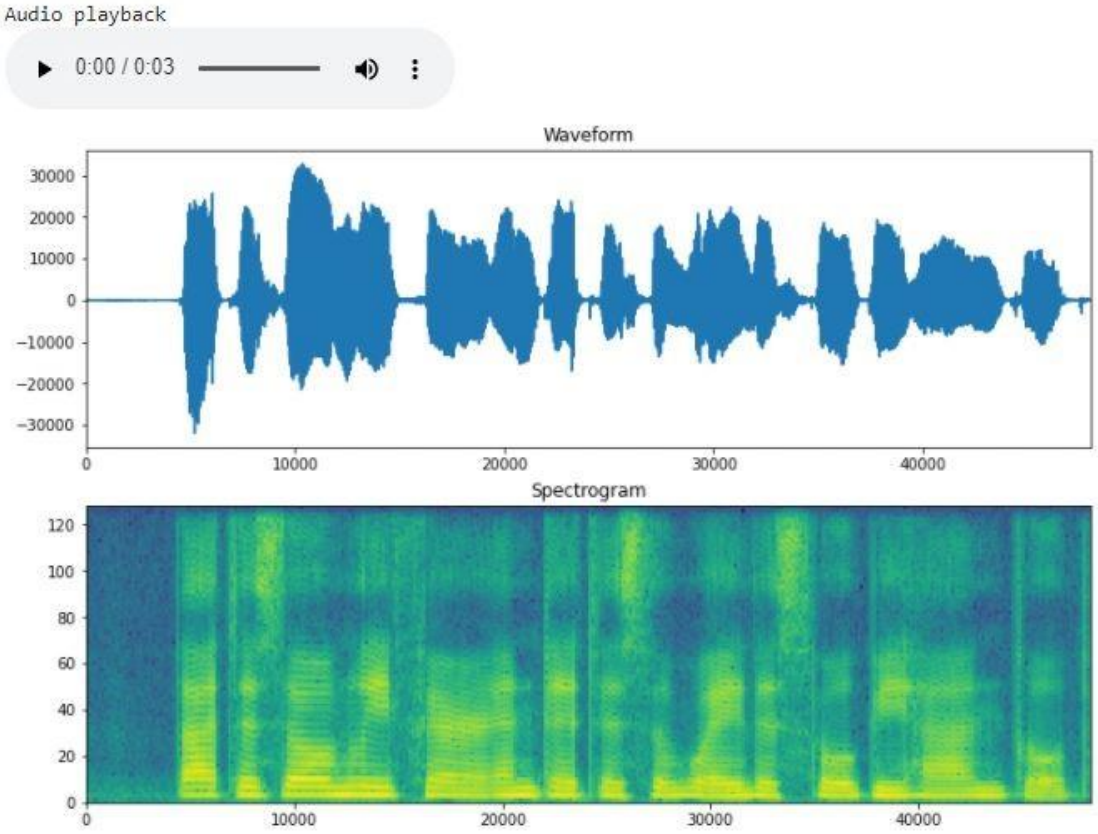


Esto se consigue abriendo cada audio de forma individual, es decir, decodificando el audio con extensión flac.

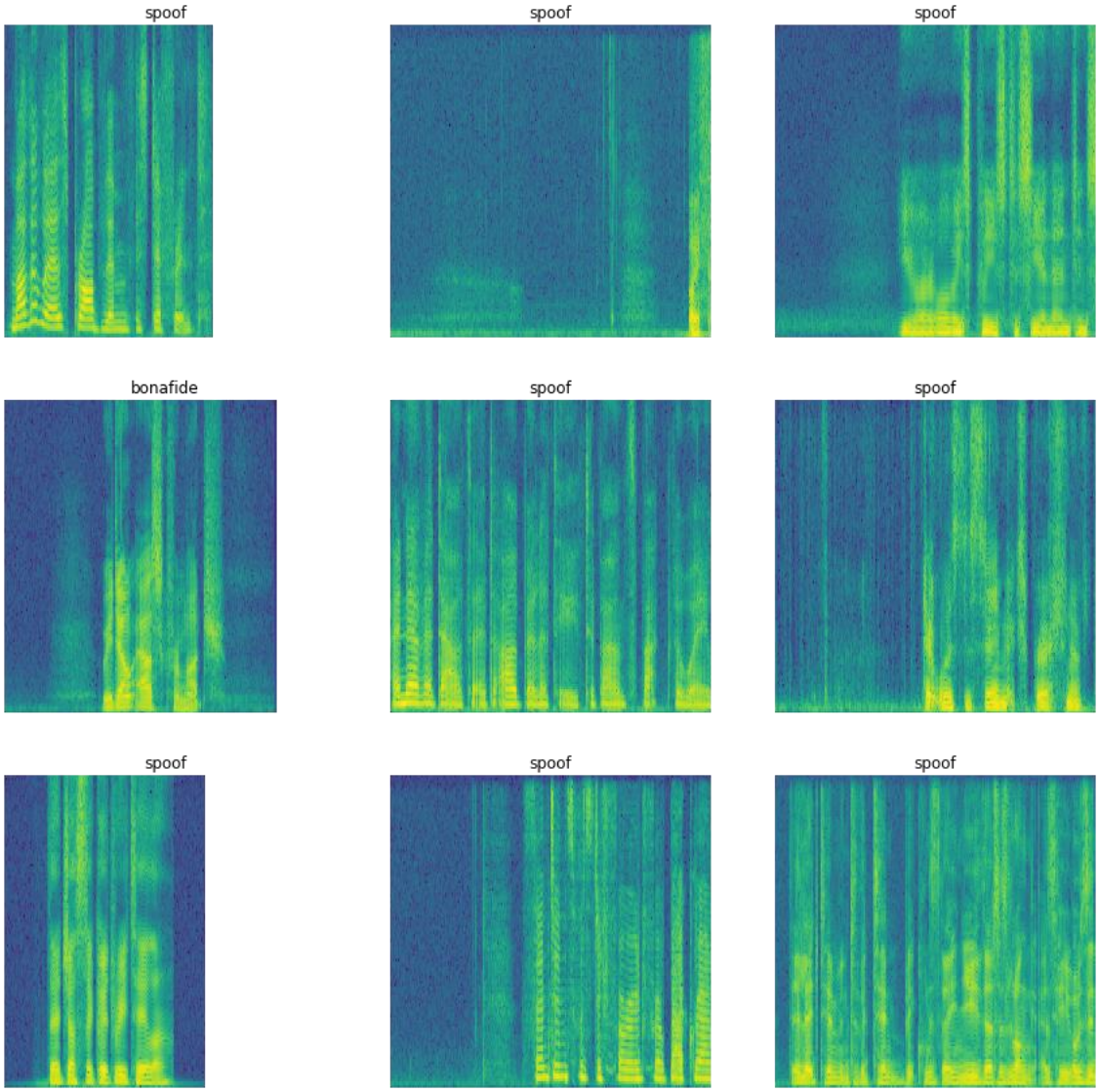
Como se puede ver en la imagen, cada *"waveform"* tiene su etiqueta asociada y tendrá una longitud de 48.000 muestras.

Una vez obtenido el *waveform* de cada audio, se puede pasar a obtener los espectrogramas, que se obtiene de hacer la transformada de Fourier de corto tiempo (STFT) con una longitud de la ventana de 255 muestras y 128 pasos por muestra.

A continuación, se muestra tanto el *waveform* como el espectrograma correspondientes a un audio de 3 segundos.



Por último, y a motivo de exposición, se muestran algunos de los espectrogramas que se han conseguido obtener a partir de distintos audios:



En definitiva, este es el aspecto de los espectrogramas con sus respectivas etiquetas que se van a utilizar para entrenar, validar y testear el modelo de red de neuronas.

4. Tipo de red de neuronas analizadas

Para el modelo de red de neuronas se ha utilizado el que se define por defecto para reconocimiento de audios simples de los tutoriales que tiene TensorFlow en su página, adaptada a nuestra práctica.

Este modelo de red de neuronas se basa en un modelo de red neuronal convolucional (CNN). Estas redes son un tipode de redes donde las "neuronas" corresponden a campos receptivos de una manera muy similar al de las neuronas en la corteza visual primaria de un cerebro biológico. Este tipo de redes son muy efectivas para tareas clasificación y segmentación de imágenes.

La razón por la que se utiliza esta red neuronal, y por la que nosotros también hemos decidido utilizarla, es porque estamos convirtiendo los ficheros de audio (con extensión flac) a imágenes (obteniendo sus correspondientes espectrogramas), por lo que será nuestra primera opción para la clasificación de los audios de cada locutor.

5. Configuración de los experimentos

Como se ha mencionado en capítulos anteriores, nos encontramos con dos modelos iguales a los que se les pasará un grupo de entrenamiento distinto. Antes de llegar al entrenamiento del modelo, tendremos que sacar los espectrogramas de todos los audios que utilizaremos en el proceso, es decir, los dos grupos de casos de entrenamiento, el grupo de casos de validación y el grupo de casos de test.

Tras esto, definiremos el modelo a utilizar donde, en primer instancia, se harán uso de las operaciones de "*caché"* y "*prefetch*" sobre todos los casos de validación y de entrenamiento correspondientes a cada modelo, para reducir la latencia de lectura mientras se entrena el modelo.

En cuanto las capas utilizadas, destacaremos dos de las capas utilizadas:

- *Normalization*: que normalizará cada pixel del espectrograma.
- *Resizing*: que reducirá la resolución de la entrada y permitirá que el modelo entrene más rápido.

```
Input shape: (374, 129, 1)
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
resizing (Resizing)          (None, 32, 32, 1)         0
_________________________________________________________________
normalization (Normalization (None, 32, 32, 1)         3
_________________________________________________________________
conv2d (Conv2D)              (None, 30, 30, 32)        320
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 28, 28, 64)        18496
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 14, 14, 64)        0
_________________________________________________________________
dropout (Dropout)            (None, 14, 14, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 12544)             0
_________________________________________________________________
dense (Dense)                (None, 128)               1605760
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 258
=================================================================
Total params: 1,624,837
Trainable params: 1,624,834
Non-trainable params: 3
_________________________________________________________________
```

En esta imagen se puede ver todas las capas que tiene el modelo. Como se dijo en el primer capítulo, el segundo modelo será idéntico al primero, por lo que estará compuesto de las mismas capas.

En cuanto a los parámetros que le hemos pasado al modelo, cabe destacar que hemos definido ***seis épocas*** para entrenarlo, ya que, si utilizábamos más épocas, solía detenerse el entrenamiento. La primer razón era que utilizamos en parámetro *"callback"* un "*EarlyStopping*", donde definíamos que la "paciencia" del entrenamiento se estableciera a dos épocas, es decir, que si tras dos épocas de entrenamiento esta no arroja mejores resultados, su ejecución se detiene. La segunda razón es que el entrenamiento se detenía de manera abrupta porque consumía todos los recursos que facilita el entorno.

6. Resultados obtenidos

Partiendo de que se utilizan seis etapas para entrenar cada modelo de red de neuronas, pasaremos a mostrar los resultados que se obtienen de ejecutar cada modelo por separado.

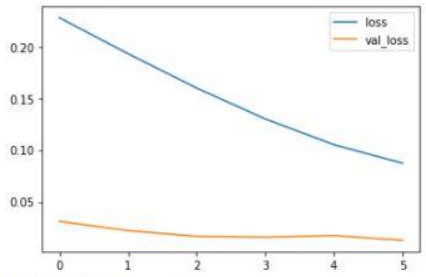
6.1. Primer modelo

El primer modelo, que tiene más objetos de prueba, tarda en ejecutar 1.184 segundos en total (20 minutos aproximadamente).

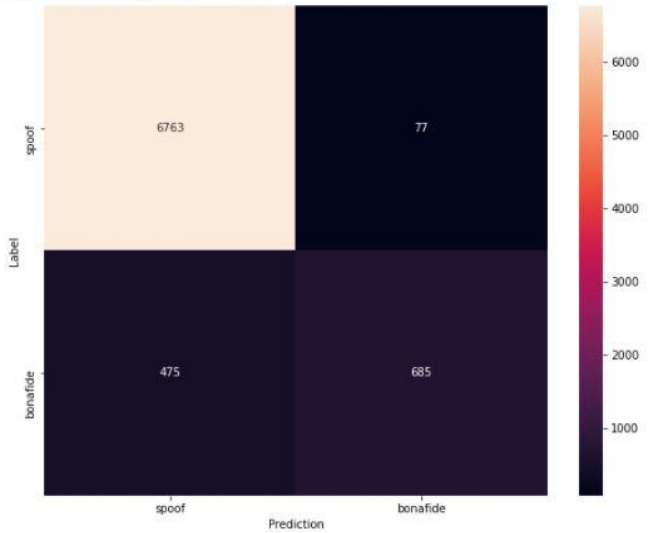
```
Epoch 1/6
625/625 [==============================] - 501s 799ms/step - loss: 0.2548 - accuracy: 0.9002 - val_loss: 0.0309 - val_accuracy: 0.9923
Epoch 2/6
625/625 [==============================] - 132s 211ms/step - loss: 0.2042 - accuracy: 0.9094 - val_loss: 0.0220 - val_accuracy: 0.9920
Epoch 3/6
625/625 [==============================] - 132s 212ms/step - loss: 0.1718 - accuracy: 0.9237 - val_loss: 0.0165 - val_accuracy: 0.9923
Epoch 4/6
625/625 [==============================] - 139s 223ms/step - loss: 0.1427 - accuracy: 0.9372 - val_loss: 0.0157 - val_accuracy: 0.9935
Epoch 5/6
625/625 [==============================] - 142s 228ms/step - loss: 0.1136 - accuracy: 0.9528 - val_loss: 0.0172 - val_accuracy: 0.9933
Epoch 6/6
625/625 [==============================] - 138s 221ms/step - loss: 0.0923 - accuracy: 0.9636 - val_loss: 0.0126 - val_accuracy: 0.9935
```

También se ha obtenido la gráfica donde se refleja las curvas de perdida de entrenamiento y validación, para observar cómo ha mejorado el modelo durante el entrenamiento. Adicionalmente, se incluirá la matriz de confusión, que es útil para ver qué tan bien funcionó el modelo agrupando los audios en cada una de sus etiquetas.

A continuación, se insertan ambas imágenes:



Test set accuracy: 93%



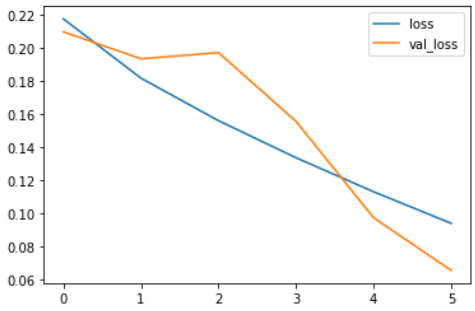
En la matriz de confusión se muestran cómo se han predicho las distintas etiquetas y cuáles eran sus verdaderas etiquetas. En ella, podemos apreciar que los valores que caen en la diagonal principal son los que se han predicho la etiqueta de manera correcta. Y los de la diagonal secundaria son los que han predicho la etiqueta de manera errónea.

6.2. Segundo modelo

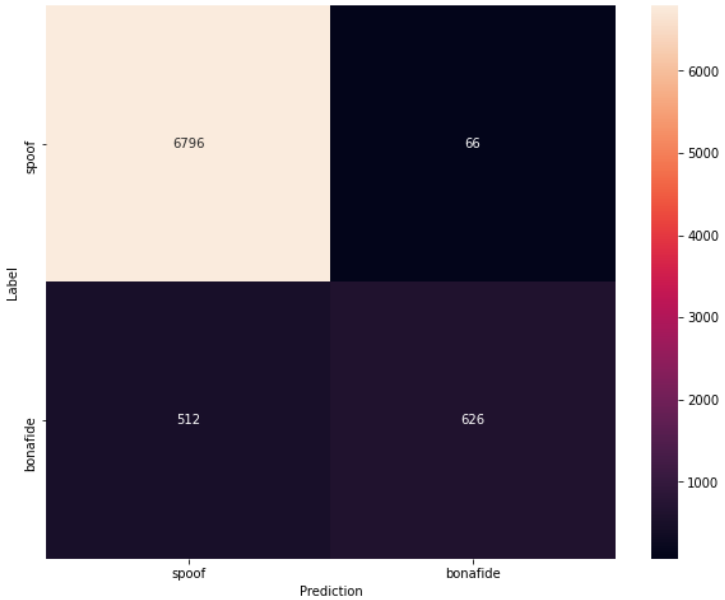
Por lo que respecta al segundo modelo, que tendrá el resto de los casos, indicado en apartados anteriores, tiene una duración total de 2.336 segundos (40 minutos aproximadamente).

```
Epoch 1/6
524/524 [==============================] - 395s 752ms/step - loss: 0.2410 - accuracy: 0.8959 - val_loss: 0.2100 - val_accuracy: 0.9477
Epoch 2/6
524/524 [==============================] - 384s 733ms/step - loss: 0.1877 - accuracy: 0.9087 - val_loss: 0.1937 - val_accuracy: 0.9252
Epoch 3/6
524/524 [==============================] - 389s 742ms/step - loss: 0.1583 - accuracy: 0.9187 - val_loss: 0.1974 - val_accuracy: 0.9022
Epoch 4/6
524/524 [==============================] - 382s 729ms/step - loss: 0.1374 - accuracy: 0.9369 - val_loss: 0.1558 - val_accuracy: 0.9287
Epoch 5/6
524/524 [==============================] - 392s 747ms/step - loss: 0.1174 - accuracy: 0.9507 - val_loss: 0.0975 - val_accuracy: 0.9600
Epoch 6/6
524/524 [==============================] - 394s 751ms/step - loss: 0.0979 - accuracy: 0.9596 - val_loss: 0.0657 - val_accuracy: 0.9758
```

Además, como en el primer modelo, se han obtenido tanto la gráfica que refleja las curvas de pérdida de entrenamiento y validación, como la matriz de confusión de este segundo modelo. A continuación se pasan a insertar ambas imágenes:



Test set accuracy: 93%



En la matriz de confusión se muestran cómo se han predicho las distintas etiquetas y cuáles eran sus verdaderas etiquetas. En ella, podemos apreciar que los valores que caen en la diagonal principal son los que se han predicho la etiqueta de manera correcta. Y los de la diagonal secundaria son los que han predicho la etiqueta de manera errónea.

7. Conclusiones

Como se puede observar con las gráficas obtenidas anteriormente, el modelo de red de neuronas funciona bien para nuestro caso, con una media entre los dos modelos de un 93% de acierto con el conjunto de casos de validación.

El mayor problema del modelo es el tiempo que tarda en realizar la ejecución del entrenamiento del modelo, en torno a los 20 minutos , con tan solo 40.000 casos de entrenamiento, por lo que si necesitáramos un modelo de red de neuronas que contara con más casos de prueba o casos de prueba con mayor número de muestras, necesitaríamos una herramienta más potente o una mayor optimización del modelo para que el tiempo que se necesita para ejecutar el entrenamiento del modelo, sea considerablemente más bajo.

8. Bibliografía

[1] Enlace a los ficheros utilizados en la práctica:

https://drive.upm.es/index.php/s/xFNkMwK7CsufEod

[2] Página de reconocimiento de audios simples de TensorFlow:

https://www.tensorflow.org/tutorials/audio/simple_audio

[3] Enlace al manual de TensorFlow:

https://www.tensorflow.org/guide?hl=es-419

[4] Enlace al repositorio de github (para ver el código fuente):

https://github.com/M-T3K/BiometriaVoz2K21