Verificación por voz utilizando redes neuronales

Francisco Basili
Ingeniería Electrónica
ITBA
Buenos Aires, Argentina
fbasili@itba.edu.ar

Bautista Schneeberger
Ingeniería Electrónica
ITBA
Quilmes, Argentina
bschneeberger@itba.edu.ar

Kevin Wahle
Ingeniería Electrónica
ITBA
Buenos Aires, Argentina
kwahle@itba.edu.ar

Sergio Andrés Peralta
Ingeniería Electrónica
ITBA
Buenos Aires, Argentina
speralta@itba.edu.ar

***Resumen*—El objetivo de este artículo es mostrar la implementación de un programa de identificación por voz utilizando técnicas modernas de redes neuronales, llamadas ECAPA TDNN (Emphasized Channel Attention, Propagation and Aggregation), comparando entre distintos tipos de redes, y pudiendo observar una simple implementación en Python, basándose en la implementación de SpeechBrain [1], basado en Pytorch [2].**

I. Introducción

Dentro del área del procesamiento de señales, el procesamiento de audio ocupa una gran proporción del mismo. A su vez, dentro de este subgrupo, se desarrollan diferentes subtópicos como el reconocimiento de voz, la síntesis "text-to-speech", el filtrado de ruido externo y la verificación de orador. Esta última resulta útil en el área de la seguridad, donde por medio de ella podríamos desbloquear o acceder a ciertos dispositivos o información. Cabe destacar que el objetivo de la verificación de orador no es identificar quién está hablando ni qué está diciendo, sino que solo se busca, mediante un análisis del espectro del audio, comparar si el orador de dos audios distintos es el mismo o no.

II. Conocimientos previos

Una red neuronal, se puede pensar como una sucesiva capa de filtros, que permite obtener distintas representaciones del objeto a clasificar. Cada capa tiene asociado una serie de pesos, el proceso de aprendizaje consta de encontrar los pesos óptimos. La *loss function*, mide el error entre la predicción y el *target* verdadero (véase fig 2). Esto se realimenta para reajustar los pesos, proceso llamado *backpropagation*. En un principio se comienza con valores aleatorios de pesos, pero luego se *entrena* la red hasta minimizar el error. [3]

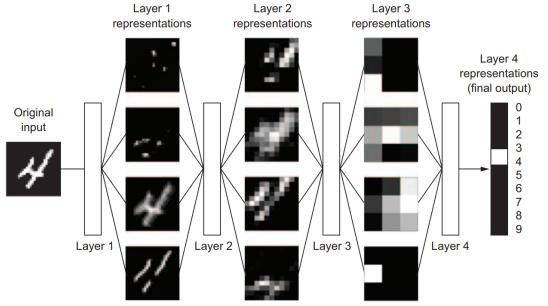

Figura 1: Distintas *layers* para identificación del número 4

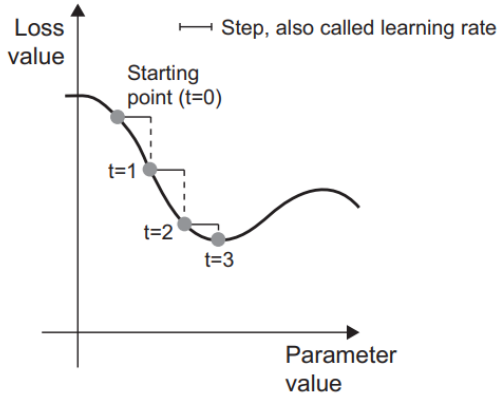

Figura 2: *Loss function* y su optimización en varios steps

La optimización basada en gradiente, es una técnica usada para inferir el ajuste de los coeficientes en el entrenamiento. Permite conocer que tanto hay que cambiar los coeficientes. Dado que en general son miles o millones, no es trivial encontrar una expresión analítica del gradiente, por lo que se aproxima. Véase la fig 3.

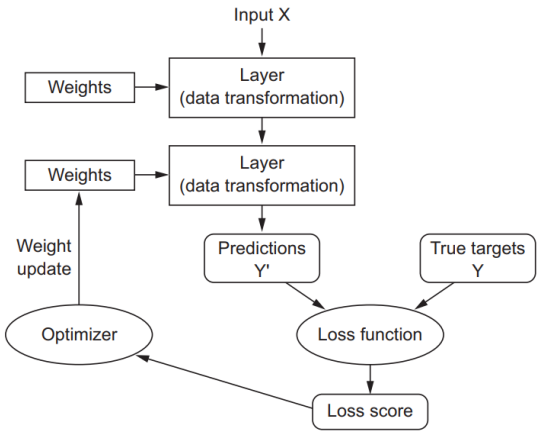

Figura 3: Esquema de una red neuronal

III. Arquitectura

El código en *SpeechBrain* sigue una arquitectura modular e independiente, la cual facilita la modificación en algunos

de sus aspectos, tales como carga de datos, decodificación, procesamiento de señales, entre otros.

SpeechBrain se puede describir como una *Toolkit*, y se le debe definir un modelo de entrenamiento. Este loop de entrenamiento se puede observar resumido en la fig 4. [1]

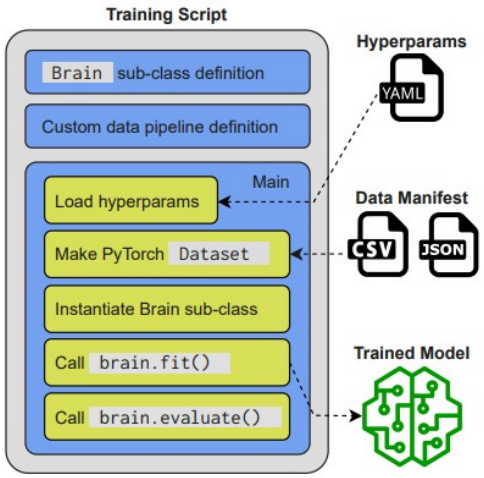

Figura 4: Script de Entrenamiento

Como se puede observar el entrenamiento comienza cargando los *hyperparameters*, los cuales se encuentran en formato YAML que son fácilmente legibles. *SpeechBrain* inicializa las clases automáticamente con el YAML, con las especificaciones dadas. Algunas de las especificaciones son el número de neuronas, el formato de input, samplerate y tipo de modelo a utilizar (ENCAPA-TDNN en nuestro caso). En la sección IX-A se puede observar un ejemplo de aplicación reducido.

Luego se generará un archivo de formato CSV o JSON con la metadata del input. En el caso de *speech recognition*, este archivo contendría las palabras que se dijeron en el respectivo audio con su path.

Las secuencias de voz pueden variar en longitud, y requieren de relleno con ceros para su comparación a igual longitud, con cierta normalización. Esto trae consigo otra dificultad que consiste en la optimización de recursos computacionales en lo que respecta a este tipo de normalización. *SpeechBrain* permite seleccionar entre distintas estrategias de procesamiento para obtener el código mas óptimo.

Dentro de las sub-clases de *Brain*, se encuentra, entre otros, el método *Brain.fit()*, en el cual se entrena un modelo basado en redes neuronales. Se definen predicciones dependiendo de ciertas entradas, y posibles errores. Aquí se determina el Phonema Error Rate (PER %), el cual varía dependiendo de los diferentes modelos utilizados.

IV. Utilización de Neural Networks

IV-A. TDNN

Inicialmente se empleaba el uso de HMM ("Hidden Markov Models") para poder comparar entre piezas de audio por su versatilidad para excitaciones con desplazamientos en tiempo. Luego se diseñaron las TDNN ("Time Delayed Neural Network") y se demostró que generaban mejores resultados que las anteriores [4].

Las TDNN son redes neuronales en donde cada neurona, además de recibir la salida de las neuronas anteriores en el tiempo actual, recibe el resultado de dichas neuronas en tiempos anteriores o posteriores (véase fig. 5). Esto le permite relacionar los coeficientes espectrales con cierto desplazamiento temporal (sin importar el momento exacto en el que hayan ocurrido) y así obtener conclusiones más generales y mejores (véase fig. 6). Por ejemplo, una TDNN que reconozca con qué letra empieza una palabra podría aprender que luego de emitida la primer letra, el peso que tendrán los instantes posteriores debería ser mucho menor. [4]

Para entrenarla se utiliza el método de "Backpropagation" [5] insertando copias de cada señal de entrada desfasadas temporalmente. Luego, se suman los gradientes para cada instante y se modifica el peso de la arista con el promedio de los gradientes como dice el método. El éxito de este tipo de red se basa en entrenarla con muestras iguales pero desfasadas en el tiempo y así lograr la característica de tiempo invariante.

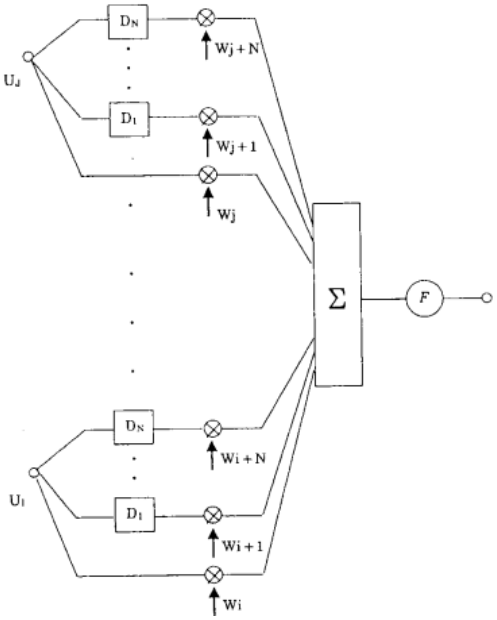

Figura 5: Desfasaje internos de las inputs

IV-B. ECAPA-TDNN

En su aporte *SpeechBrain* realiza una serie de mejoras sobre las TDNN llamado ECAPA-TDNN ("Emphasized Channel Attention, Propagation and Aggregation") en donde:

- Permiten que las neuronas se salten capas afectando directamente a otras neuronas que se encuentran 2 o más capas más adelante.
- Se desarrolla una fórmula matemática para potenciar el énfasis de ciertas aristas (lo que su nombre indica como "Emphasized Channel Attention") y permitiendo disminuir la cantidad de aristas poco influyentes.

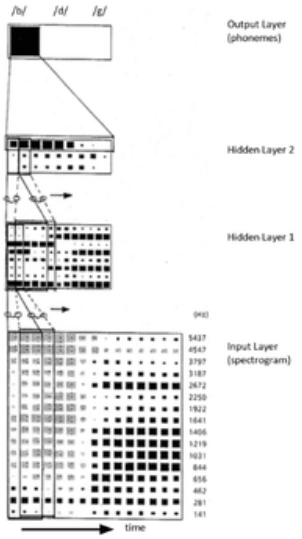

Figura 6: Inputs y activación de neuronas

En su paper [6] se demuestra los beneficios de esta mejora tanto en el desempeño de la red como en el incremento de la velocidad de entrenamiento de la misma.

V. Cosine Similarity

A diferencia de otras redes neuronales, *SpeechBrain* no utiliza la distancia Euclidiana para determinar la diferencia entre los vectores que representan a los audios. En su lugar, utiliza la distancia cosenoidal definida como:

$$CDF(v_1, v_2) = \frac{v_1 \cdot v_2}{\|v_1\|\|v_2\|} \tag{1}$$

Como se puede ver en eq. (1) y en fig. 7, la distancia cosenoidal se centra en comparar la diferencia angular entre dos vectores y no la magnitud de la distancia entre ellos. Esto es sumamente necesario en aplicaciones sobre lenguaje en donde la diferencia entre las longitudes de dos vectores puede distorsionar la medición de la distancia haciendo que parezcan más diferentes de lo que realmente son [7]. Por ejemplo, un audio de 5 minutos va a tener una gran distancia euclidiana con uno de 5 segundos por el mero hecho de que uno tiene mucha más información que el otro, mientras que con la distancia cosenoidal esto no sucede. Una vez que el audio original del orador para comparar ya haya sido cargado, por cada audio a comparar se detectará qué dice el orador y con la red neuronal se buscará generar el vector que represente dicha frase en base a las características del orador original. Por otro lado, se obtendrán los parámetros propios del nuevo audio y con la distancia cosenoidal se buscará estimar cuan similares son.

VI. Dataset de entrenamiento

El desempeño de toda red neuronal depende en gran medida por la calidad de los datos con los que se entrena a la misma. En este caso se utiliza el *VoxCeleb2* dataset el cual consta de más de 1000 vídeos de Youtube de famosos de ambos

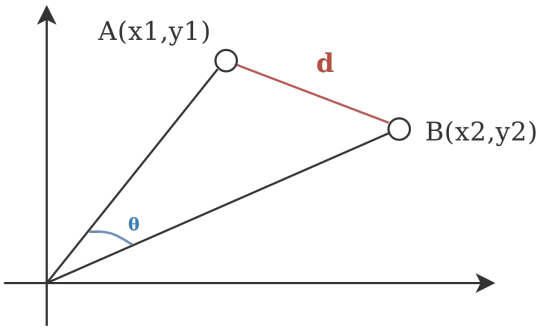

Figura 7: Diferencias distancia euclidiana y cosenoidal

géneros y de más de 60 nacionalidades hablando en distintos idiomas y en entornos con distorsiones variadas [8]. Esto permite que el entrenamiento sea robusto y la red pueda predecir correctamente con audios que no estén grabados específicamente en determinados idiomas ni en condiciones muy restrictivas con bajo ruido.

VII. Implementación

Se implementó en python una GUI que le permite al usuario seleccionar diferentes archivos de audios para compararlos entre sí y ver si realmente pertenecen a la misma persona o no. Para ello se normalizan los audios:

- Se verifica la extensión del archivo y se hace la conversión pertinente.
- Se convierten a mono.
- Se realiza un resampleo con una frecuencia normalizada de ser necesario.

Luego se pueden elegir los diferentes audios a comparar tan solo arrastrándolos hacia el número de selección de audio pertinente. Luego se presiona en el botón para comparar y comienza a funcionar el algoritmo mencionado en la sección III y desarrollado en la secciones IV y V. Se obtiene como resultado un cálculo que representa cuán similares son 2 voces y una conclusión final de si son de la misma persona o no.

Finalmente se puede observar como los audios se pueden subir desde la PC o se pueden grabar en el momento seleccionando el icono con el micrófono. En la fig. 8 se puede observar una imagen ilustrativa de la GUI en funcionamiento y en en la sección IX-B se puede observar el código del back-end utilizado. El código posee dos funciones, una realiza el acondicionamiento de la señal y la otra posee el algoritmo que realiza la verificación.

VIII. Conclusiones y futuros estudios

Se comprobó la efectividad de la red neuronal ECAPA-TDNN, realizando diferentes pruebas con voces propias y de diferentes sitios. Se pudo observar que se puede realizar una distinción realmente efectiva con tan solo unos pocos segundos de audio (menos que 5 segundos).

Sin embargó se notó que entre algunas voces la diferencia era poca notoria, lo cual llevó al aumento del umbral de

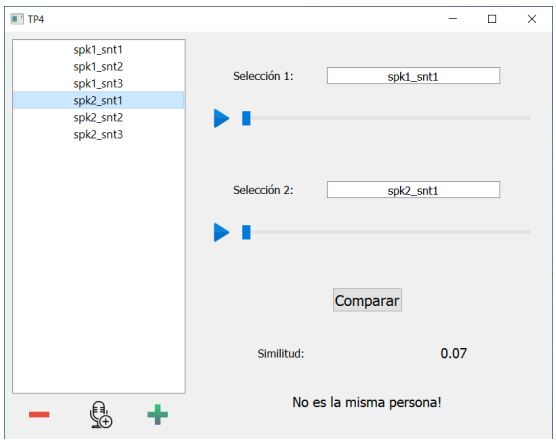

Figura 8: Interfaz de la aplicación

decisión de si se trataba de la misma persona o no. Esto llevó a la conclusión de que no sería efectivo utilizarlo de esta manera para seguridad de algo valioso, aunque podría ser utilizado para simple autenticación con cierto grado de compromiso. Para aumentar la fiabilidad, se debería entrenar al modelo con audios de la persona en cuestión, lo cual no se realizó en este caso con el objetivo de darle una mayor generalidad.

Las aplicaciones sobre verificación de voz en seguridad son muy amplias, lo cual permite focalizar futuros estudios en cualquiera de éstas. Hoy en día uno cuenta con cientos de cuentas pertenecientes a diferentes páginas o redes sociales, cada una con su respectiva contraseña escritas. Es por esto que muchos optan por otro tipos de protección, ya sea con huella digital, reconocimiento facial, o identificación por voz. Es por ello que se decidió llevar los futuros estudios a implementación de encriptación y desencriptación de archivos mediante verificación por voz, en lugar de utilizar contraseñas ordinarias.

Referencias

[1] M. Ravanelli, T. Parcollet, P. Plantinga, A. Rouhe, S. Cornell, L. Lugosch, C. Subakan, N. Dawalatabad, A. Heba, J. Zhong, J.-C. Chou, S.-L. Yeh, S.-W. Fu, C.-F. Liao, E. Rastorgueva, F. Grondin, W. Aris, H. Na, Y. Gao, and Y. Bengio, "Speechbrain: A general-purpose speech toolkit," 06 2021.

[2] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[3] F. Chollet, *Deep Learning with Python*. Manning, Nov. 2017.

[4] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. Lang, "Phoneme recognition using time-delay neural networks," *Acoustics, Speech and Signal Processing, IEEE Transactions on*, vol. 37, pp. 328 – 339, 04 1989.

[5] K. Lang, A. Waibel, and G. Hinton, "A time-delay neural network architecture for isolated word recognition," *Neural Networks*, vol. 3, pp. 23–43, 12 1990.

[6] B. Desplanques, J. Thienpondt, and K. Demuynck, "Ecapa-tdnn: Emphasized channel attention, propagation and aggregation in tdnn based speaker verification," 10 2020.

[7] M. D. Balasingam and C. S. Kumar, "Refining cosine distance features for robust speaker verification," in *2018 International Conference on Communication and Signal Processing (ICCSP)*, 04 2018, pp. 0152–0155.

[8] J. S. Chung, A. Nagrani, and A. Zisserman, "Voxceleb2: Deep speaker recognition," 09 2018, pp. 1086–1090.

IX. Anexo

IX-A. Ejemplo de YAML

```yaml
# Output parameters
out_n_neurons: 7205

# Model params
compute_features:
!new:speechbrain.lobes.features.Fbank
n_mels: !ref <n_mels>

embedding_model:
!new:speechbrain.lobes.models.ECAPA_TDNN.ECAPA_TDNN
input_size: !ref <n_mels>
channels: [1024, 1024, 1024, 1024, 3072]
kernel_sizes: [5, 3, 3, 3, 1]
dilations: [1, 2, 3, 4, 1]
attention_channels: 128
lin_neurons: 192

classifier:
!new:speechbrain.lobes.models.ECAPA_TDNN.Classifier
input_size: 192
out_neurons: !ref <out_n_neurons>
```

IX-B. Implementación en Python

```python
import torchaudio
from speechbrain.pretrained import SpeakerRecognition

# Carga de archivo y normalizacion
def loadFile(filePath):
    fs_norm = 16000     # Frecuencia de sampleo necesaria para hacer la comparacion

    signal, fs = torchaudio.load(filePath)

    # Conversion a mono
    if len(signal.shape) > 1:
        signal = signal.mean(axis=0)

    # Si es necesario se realiza resampleo
    # Utiliza un filtro FIR de orden 6 con ventana de Hann
    if fs != fs_norm:
        signal = torchaudio.transforms.Resample(fs, fs_norm)(signal)

    return signal

# Comparacion de audios para determinar si son de la misma persona
def verifySpeaker(audio1, audio2):

    #ECAPA-TDNN Model
    verification = SpeakerRecognition.from_hparams(source="speechbrain/spkrec-ecapa-voxceleb", savedir="pretrained_models/spkrec-ecapa-voxceleb")

    TH = 0.5            # Limite donde se consideran iguales

    verify = lambda x, y: verification.verify_batch(x, y, threshold=TH)

    score, prediction = verify(audio1, audio2)

    score = score.item()
    prediction = prediction.item()

    return score, prediction
```