DAS-PEAK

Voice biometrics service

1. das-Peak microservice

das-Peak is intended for text-independent speaker verification/recognition. More in particular, das-Peak determines the similarity between two audio recordings (in terms of the speakers present in them) previously stored in *WAV* format. These data can be extracted from any of the following sources:

- Veridas Voice Capture SDK's: Mobile
- Audio recorded from any input in *WAV* format

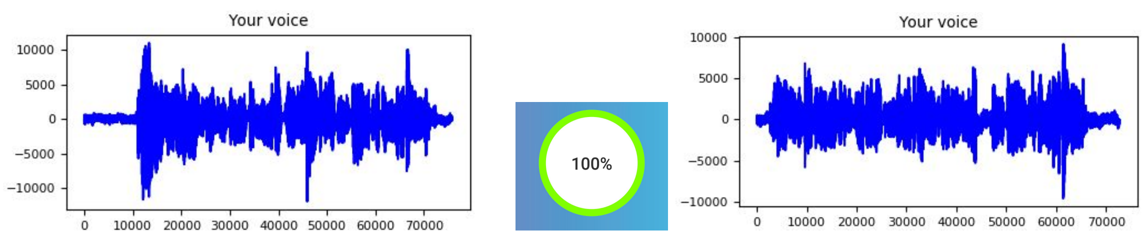
Within the voice biometrics field, two scenarios are typically handled:

- **Verification:** The process of checking the identity of a person by comparing two audio recordings.
- **Identification:** The process of searching a person or a set of persons within a database of identities and its audio input data.

So far, we hold solution for the verification problem.

1.1. Verification

- **Speaker Verification (1:1):** Given two audio recordings, the system returns a score based on the similarity of both of them, not regarding to speech recognition but to the speakers present in them.

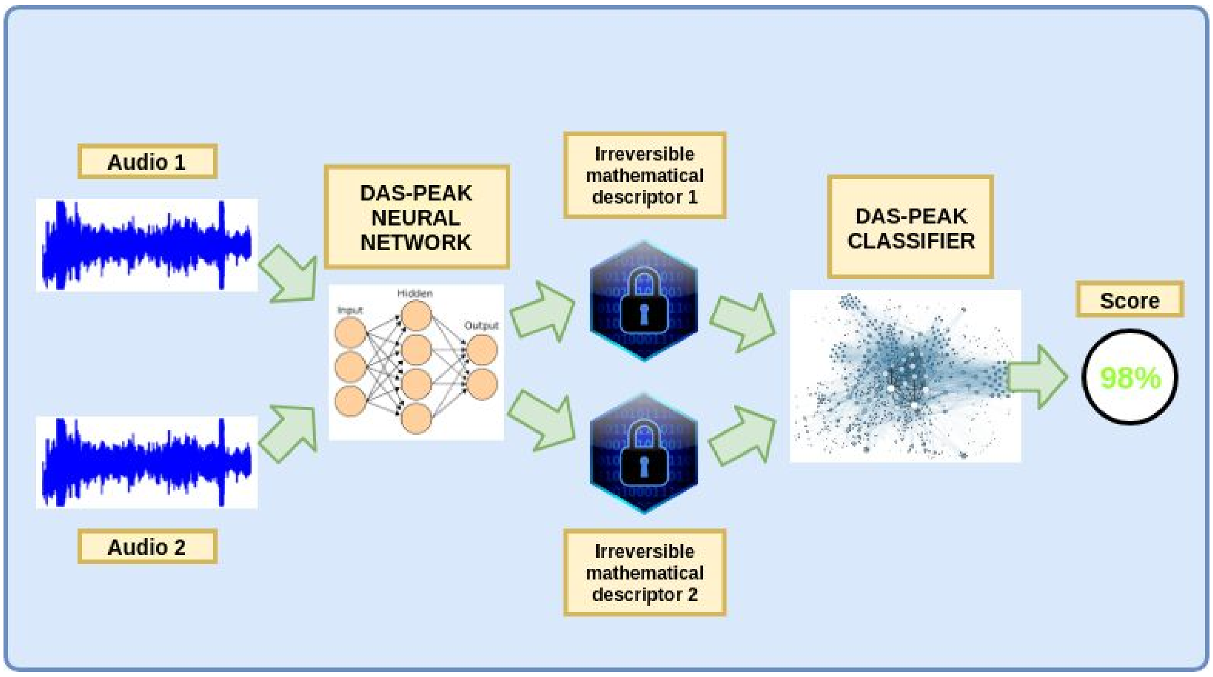


2. How does it work?

2.1. Biometric engine

This microservice is offered as an API_REST format:

- Two audio recordings are sent to the server.
- The audio recordings are pre-processed (i.e., speech features are extracted), estimated their SNRs (*Signal-to-Noise Ratios*) and performed a VAD (*Voice Activity Detection*) analysis.
- The audio recordings are converted into irreversible mathematical descriptors.
- Both mathematical descriptors are compared using Veridas' similarity estimation algorithms. The working point can be controlled to set a particular FPR (*False Positive Rate*) as well as a particular FNR (*False Negative Rate*).
- A matching score is provided.
- All the user information (i.e., both the audio recordings as well as the processed data) is **immediately deleted**.



3. System performance

das-Peak is the **proprietary** speech biometrics engine developed by Veridas, using deep learning and other AI (*Artificial Intelligence*) technologies.

This system has been tested over the NIST SRE (*Speaker Recognition Evaluation*) evaluations of 2012 and 2016. NIST SRE results have been verified by ViVoLab (Voice Input Voice Output Laboratory) of the University of Zaragoza (Spain). The primary performance measure for these evaluations was a detection cost, defined as a weighted sum of miss and false alarm error probabilities.

The results obtained are depicted in the following table:

NIST 2012	**Act DCF known**	**Min DCF known**	**EER known**	**Act DCF unknown**	**Min DCF unknown**	**EER unknown**
Interview with no added noise: Cc1	0.4349	0.3613	6.23%	0.4349	0.3162	5.60%
Telephone with no added noise: Cc2	0.4911	0.4166	4.50%	0.4867	0.3850	4.15%
Interview with added noise: Cc3	0.4054	0.3530	6.64%	0.4054	0.3494	6.75%
Telephone with added noise: Cc4	0.7201	0.6276	7.78%	0.7134	0.4504	6.85%
Telephone in noisy environments: Cc5	0.5431	0.4640	5.05%	0.5376	0.3183	4.24%

NIST 2016	**Act Cprim eq.**	**Min Cprim eq.**	**EER**	**Act Cprim uneq.**	**Min Cprim uneq.**	**EER**
Open condition	0.6213	0.6072	8.66%	0.6785	0.6065	8.50%

The computational times to create a mathematical descriptor vector are:

- If ran on CPU: 2.7 seconds
- If ran on GPU: 2.7 seconds

The computational times to compare two mathematical descriptor vectors are:

- If ran on CPU: 0.4 seconds
- If ran on GPU: 0.4 seconds

Annex 1: API operation

Requests to the server can be in application/json, and some endpoints accept multipart/form-data as well. Responses are always JSON and, in case of a failure, the server response is a JSON with the following fields:

Field	Required	Description
exception	yes	Error code, for example: *AudioFormatError*, *SamplingRateError*, etc.
message	no	A message providing more information about what went wrong.

Example:

```
{
        "exception": "BitsPerSampleError",
        "message": "The number of bits per sample of the anchor input audio file is 8, but 16 is
                    required."
}
```

1.1. API endpoints

This microservice exposes an API with the following features.

1.1.1 Audio similarity between two audio recordings

The microservice receives a POST request with two audio files containing raw data in *WAV* format each, and returns a similarity value indicating how similar are both speakers.

POST /similarity/one2one

Request:

Content-Type: application/json, multipart/form-data

Name	Req.	Type	Description
anchorAudio	yes	file (*WAV*)	As multipart/form-data, it should be a file, as application/json, the file content encoded in base64.
targetAudio	yes	file (*WAV*)	As multipart/form-data, it should be a file, as application/json, the file content encoded in base64.

Response:

Content-type: application/json

Returns the confidence (or similarity) between both audio recordings, so the closer this number to 1 the greater the similarity between both audio recordings and vice versa (i.e., the closer this number to 0 the less the similarity between both audio recordings). In other words, this number is within the range [0, 1].

```
{
	“confidenceNumber”: 0.9999999975560285
}
```

Errors:

Code	HTTP Status	Message
UnexpectedError	400	An unknown error has occured while processing the request.
AudioFormatError	400	The input file is not a standard WAV file as required.
SamplingRateError	400	The sampling rate of the input audio file is A, but a sampling rate of B is required.
BitsPerSampleError	400	The number of bits per sample of the input audio file is A, but B are required.
NumberOfChannelsError	400	The number of channels of the input audio file is A, but B is required.
VoiceNotDetectedError	400	No voice has been detected in this audio.

SnrThresholdError	400	The SNR is too low for this audio.

1.1.2 Is alive

The microservice receives a GET request with no parameters, and returns a 200 status code indicating that the server is up.

GET /alive

Errors:

Code	**HTTP Status**	**Message**
UnexpectedError	400	An unknown error has occured while processing the request.