# I/ Identification de locuteur

## Introduction / Titre à déterminer

Le cerveau humain est bien fait, lorsque nous entendons une voix (déjà entendu auparavant), il nous est très simple d’attribuer un visage, ou du moins une identité à cette voix. Ce réflexe automatique vient de nos nombreuses années d’expériences de vie et nous est commun à tous (ainsi qu’à de nombreux animaux). En revanche, lorsque nous décomposons ce phénomène à une échelle scientifique, cela devient tout de suite un problème très complexe qui pour vulgariser se résume en la capacité à calculer les différences entre les phonétiques incidentelles d’une vocalisation spécifique et les représentations phonologiques abstraites des mots que contient la vocalisation[[1]](#footnote-1).

L’identification de locuteur est une partie importante de notre projet MED-IA. En effet, après avoir réussi à segmenter l’audio d’un débat entre les différents participants et séparer les voix lors de superpositions multiples présentes dans un débat, il faut réussir à attribuer un nom à ces orateurs. Alors bien qu’en tant que humains nous n’ayons pas besoin de cette technologie pour savoir qui parle lors d’un débat, nous avons pour but de faire des analyses personnalisées sur le débat pour chaque orateur. Dans ce cas, il nous faut être capable de savoir automatiquement qui a prononcé quelles paroles tout du long de l’audio, et pour cela il n’est pas question de le faire à la main.

## Jeu de données

Afin de réaliser à bien notre modèle d’identification de politiciens, Alten nous a fournis leur jeu de données constitué de 4585 fichiers audio de type .wav. Chaque audio contient un monologue de 10 secondes d’un politicien, accompagné de son nom. Etant des audio avec des monologues sans interruption et d’une bonne clarté sonore, nous avons eu accès à des données de haute qualité. En revanche, lors d’un vrai débat, les voix ne sont pas souvent aussi claires et se chevauchent très souvent à cause d’interruptions. Il faut donc noter que même si notre modèle performe bien sur les données d’Alten, en cas réel il aura un peu plus de mal avec nos audio d’overlaps séparé.

## Etat de l’art

De nos jours, pour atteindre les meilleurs résultats lors de la classification d’un fichier audio, il est crucial de respecter une certaine étape qui est l’**embedding**. En français cela se traduit littéralement par « intégration », il faut comprendre cela comme une vectorisation de l’audio. Pour aller plus loin, il est possible de représenter un audio brut directement sous forme de « vecteur de caractéristiques brutes », en effet, un fichier audio est avant tout une onde, et il nous faut être capable de la représenter numériquement pour l’exploiter.

Pour cela, il faut d’abord échantillonner notre audio qui est une onde en continu afin de pouvoir les organiser dans un vecteur. Dans notre cas, notre échantillonnage est d’une fréquence de 16 kHz ce qui veut dire que chaque seconde d’audio sera représentée par un vecteur de 16000 points.

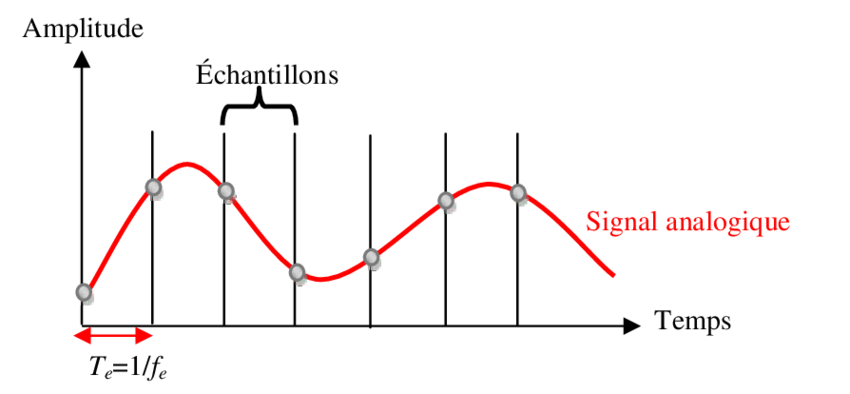


Fig1 : Echantillonage d'une onde

Ce vecteur de caractéristiques brutes en résultant est très basique, il ne capture que l’amplitude de l’onde audio à divers points dans le temps et ne tient pas compte de caractéristiques plus complexes comme la fréquence ou le timbre. En principe, il serait suffisant pour entrainer un modèle de classification de locuteurs, néanmoins, il prendrait beaucoup de temps à entrainer en raison du nombre de valeurs et ne donnerait pas des résultats satisfaisants. Une grosse partie de la réflexion autour de l’identification de locuteurs aura été autour du choix de modèle pour générer le meilleur embedding possible, qui serait capable de capturer des relations complexes et non-linéaires de l’audio. Alten utilise deux modèles alternatifs, le I-Vector qui représente un audio en utilisant un vecteur de caractéristiques statistiques se basant sur des modèles de mélange gaussien, ainsi que le X-Vector, un modèle plus moderne basé sur les réseaux neuronaux profonds (deep neural networks), et sur lequel nous reviendrons en détail plus tard.

Nous avons passé beaucoup de temps sur le modèle wav2vec 2.0[[2]](#footnote-2) développé par Facebook AI en 2020 et qui était de loin le meilleur modèle à sa sortie. Le problème de ce modèle que nous avons récupéré avec la librairie torchaudio de PyTorch est qu’il est spécialisé dans « l’Automatic Speech Recognition » (reconnaissance vocale automatique), nous induisant en erreur. En effet, nous avons cru pendant un certain temps que comme le titre peut laisser croire l’ASR était conçu pour reconnaître la parole d’un orateur. En vérité il s’agit de la transcription d’un audio en texte, expliquant pourquoi nous nous retrouvions avec des vecteurs 4D augmentés à environ 9 197 568 valeurs pour un simple audio de 10 secondes. C’est d’autant plus logique car la transcription en texte requiert beaucoup plus d’informations sur l’audio que pour effectuer une détection de locuteurs. Comme vous pouvez vous en douter, nos ordinateurs n’étaient pas assez puissants pour supporter une telle quantité de donnée et il était impossible de continuer sur cette piste (nous sommes même allés jusqu’à demander un accès à une machine virtuelle d’Alten pour faire tourner le modèle mais il y a eu des problèmes d’accès).

**Paragraphe de Shawn sur NEMO**

Fort de notre succès avec la librairie Pyannote dans les domaines de la diarisation, nous avons une nouvelle fois sollicité leur expertise pour trouver un modèle d’embedding audio. Et en effet, ils en avaient bien un à proposer le modèle « embeddings ». Spécialisé dans l’embedding d’audio pour la classification (que ce soit de locuteurs, d’émotions, de langues …), ce modèle propose un embedding très léger et d’une performance inégalé grâce à la fusion de deux modèles différents, le SincNet et le X-Vector.

## Pyannote Embedding

Le modèle embeddings de pyannote est un modèle d’apprentissage profond (deep learning), écrit avec la librairie PyTorch et basé sur une architecture appelée XVectorSincNet. Pour tout audio traité avec ce modèle, nous avons en sortie un vecteur 1D avec seulement 512 valeurs. Nous allons voir maintenant comment il est possible d’avoir si peu de valeurs en résultat et atteindre des performances au sommet de l’état de l’art en matière de détection de locuteurs.

### SincNet

Le principal atout différenciant notre modèle de celui utilisé par Alten est l’utilisation d’un modèle SincNet[[3]](#footnote-3). Ce modèle est basé sur des fonctions Sinc, conçus spécifiquement pour s’adapter aux signaux audios, et qui implémentent des filtres passe-bande. Ce sont des dispositifs laissant seulement passer les fréquences d’un certain intervalle compris entre une fréquence de coupure basse et une fréquence de coupure haute. En physique, lorsque nous appliquons une transformée de Fourier à une onde nous la transformons d’un domaine temporel à un domaine fréquentiel. Cette opération mathématique nous permet de visualiser de manière plus intuitive ainsi que d’analyser les différentes fréquences qui composent le signal sonore.

Une image contenant texte, Police, nombre, diagramme

Description générée automatiquement

Fig2 : Fonction Sinc - Domaine Temporel & Fréquentiel

La première couche du modèle SincNet que nous appliquons en entrée à notre signal sonore, permet par un apprentissage automatique de trouver les meilleurs filtres passe-bande à appliquer, en ajustant les paramètres du filtre qui sont seulement les fréquences de coupure, c’est-à-dire le début et la fin du rectangle dans le domaine fréquentiel. Evidemment le signal sonore étant continu, il y a plusieurs filtres passe-bande à appliquer du début de l’audio jusqu’à sa fin, on appelle cela un banc de filtres. Tout cela permettra d’extraire des caractéristiques plus précises en se concentrant seulement sur des bandes de fréquences spécifiques plutôt que l’audio entier. Par exemple, un filtre pourrait avoir des fréquences de coupure correspondant à la gamme de fréquences produites par les voyelles, ou des consonnes…

Une image contenant ligne, diagramme

Description générée automatiquement

Fig3 : Représentation des filtres

Nous pouvons voir sur ces figures plusieurs filtres différents, à gauche proposé par un modèle classique et à droite par notre SincNet. En haut est leur représentation temporelle et en bas fréquentielle. Nous pouvons bien visualiser la simplicité des filtres du SincNet n’ayant pour paramètre que les fréquences de coupure, par rapport au traitement intégral du signal par un autre modèle. Par exemple, pour un signal composé de F filtres de longueur L, un modèle basique s’entraînera sur F\*L paramètres et le SincNet sur 2\*F.

### ii) X-Vector

Une fois que nous avons prétraité notre audio avec le SincNet, il est temps de réaliser notre embedding final. En conséquence nous utilisons le X-Vector[[4]](#footnote-4) qui est modèle avec une architecture de réseaux neuronaux à délai temporel (TDNN), un type de modèle d’apprentissage profond (DNN) particulièrement adapté aux signaux sonores car il prend en compte les contextes locaux de son entrée.

Une image contenant texte, capture d’écran, diagramme, Police

Description générée automatiquement

Fig4 : Diagramme du X-Vector

Comme nous pouvons le remarquer, notre vecteur de sortie du SincNet va passer par des couches d’apprentissage au niveau des frames (fenêtres locales) puis par une couche de Statistics Pooling afin de concentrer l’information et enfin des couches au niveau du segment en résultant, et nous pourrons récupérer notre embedding à l’avant-dernière couche avant la classification car nous voulons faire notre propre modèle.

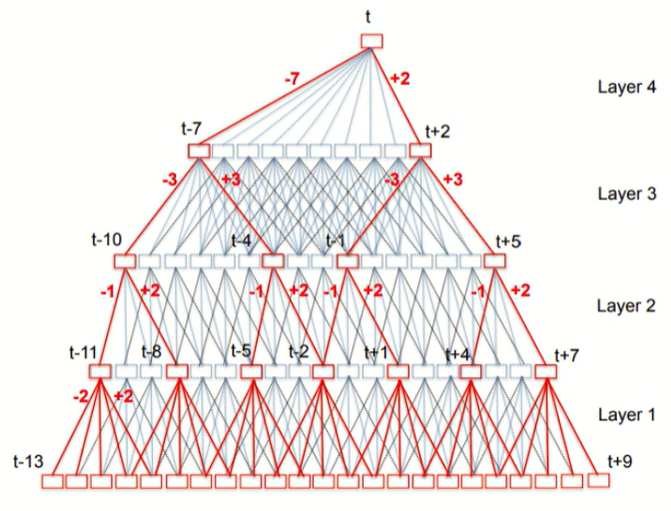


Fig6: Tableau du X-Vector

Fig5: Représentation des couches avant le Stats Pooling

Une image contenant texte, nombre, Police, capture d’écran

Description générée automatiquement

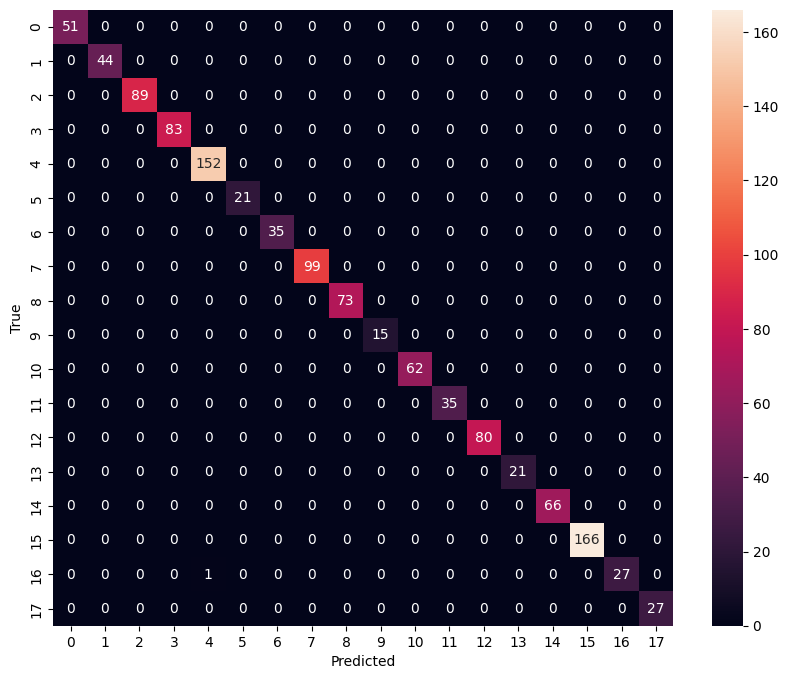
Un modèle d’apprentissage profond classique aurait prit directement toutes les valeurs du vecteur d’entrée et aurait commencé l’apprentissage des neurones sur cet ensemble. Comme nous pouvons le voir sur la figure 5, nous créons des fenêtres locales de cinq valeurs de t-2 à t+2 compris, puis nous décalons d’une valeur pour la prochaine fenêtre et ainsi de suite. Nous répétons le même principe sur les couches supérieures afin de bien apprendre les caractéristiques du signal en prenant en compte le contexte du signal, car il y a bien une suite logique à ce que quelqu’un dit à l’oral, chaque mot sont corrélés et il nous faut un modèle prenant en compte ces corrélations au long terme. (C’est le même principe que ChatGPT utilise pour comprendre ce que vous dites et écrire correctement). Comme vous pouvez constater, cela améliore énormément notre modèle comparé à un réseau de neurones classique qui prendrait toutes les valeurs de l’entrée sans tenir en compte de leur position dans la séquence. Nous pouvons constater sur la figure 6, l’évolution du contexte total de chaque fenêtre à chaque couche ainsi que du nombre de valeurs en entrée et sortie de chaque couche. On se retrouve avant la couche de Stats Pooling avec des vecteurs de 1500 valeurs au nombre T, contexte total. La couche de Stats Pooling va justement créer un vecteur moyen ainsi qu’un vecteur écart-type à partir de tous ces vecteurs, puis les combiner. Et ensuite nous avons les couches au niveau du segment pour réduire l’embedding final à seulement 512 valeurs.

## Classification par CNN

Une fois le vecteur sorti, il faut maintenant entraîner un modèle pour associer ces vecteurs d’audio avec leur locuteur. Pour cela, nous n’avons pas eu besoin d’aller chercher un modèle de classification pré-entrainé, nous avons simplement créée le nôtre. Pour ce qui est des données de type vecteur et donc de type matrice comme les audios ou bien les images, les modèles d’apprentissage profond à convolution (CNN) sont les plus adaptés. Il s’agit de simples réseaux neuronaux avec des couches de convolutions pour extraire les caractéristiques de l’entrée (à l’image d’un filtre Sinc du SincNet qui sont en réalité un type de filtre de convolution). Plusieurs filtres, dans notre cas 32, sont choisis arbitrairement par le modèle et appliqués à notre vecteur pour encore une fois l’extraction de caractéristiques. Cela peut sembler redondant avec les modèles précédents mais nous ne voulions pas prendre de risques et obtenir les meilleurs résultats possibles. Des couches de MaxPooling sont également appliqués pour réduire les sorties des filtres à leurs informations les plus importantes. Enfin nous agrégeons ces résultats, puis entrainons la sortie avec une couche de neurones classique. Nous implémentons une couche de Dropout qui va désactiver aléatoirement des neurones lors des différentes phases de l’apprentissage afin d’éviter justement le surapprentissage sur nos données d’entraînement.

## Résultats

Les résultats que nous obtenons après toutes ces opérations sont excellents, nous avons une précision de 99.91% sur notre jeu de données de test. Nous n’avons pas les chiffres exactes d’Alten mais d’après la déposante du projet il serait d’environ 85%, avec notamment des difficultés au niveau des prédictions confondant M. Macron et Mme Lepen.



Matrice de confusion du modèle sur le test set

Cette matrice de confusion représente les prédictions du modèle sur le test set par rapport à leur vraie identité. Nous pouvons voir qu’il n’y a qu’une seule erreur sur les 1147 audios, au niveau d’un audio au label 16 et prédit à 4. Cela correspond à un des audios de M. Bardella prédit comme étant de M. Mitterrand. Cela reste une erreur assez triviale car il n’existe aucune interaction entre ces deux personnes et n’en existera jamais.

Il faut bien noter que bien que ces résultats puissent paraître hors du commun, ils ne sont pas représentatifs pour un usage en cas réel. En effet les audios de très bonnes qualité comme ceux utilisés pour entrainer le modèle sont plutôt rare lors d’un débat politique. En réalité, il existe beaucoup d’interruptions de la parole et de voix chevauchantes, qu’il faut séparer avec notre modèle de séparation de voix. Bien que ce modèle fonctionne bien, les audios en résultant ne sont toujours pas d’aussi bonne qualité que ceux de base. Nous n’avons pas eu assez de temps pour créer un véritable jeu de donnée d’audios séparé par notre modèle afin de tester l’efficacité du modèle sur ce type d’audio. Sur les 23 audios séparé sur lesquels nous avons testé le modèle, 19 ont finalement été correctement prédit, correspondant à une précision de 82.61% mais il faudrait le tester plus extensivement pour faire de véritables conclusions.

1. *Human Voice Recognition Depends on Language Ability* by Tyler K. Perrachione, Stephanie N. Del Tufo, John D.E. Gabrieli [↑](#footnote-ref-1)
2. *wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations* by Alexei Baevski, Henry Zhou, Abdelrahman Mohamed, Michael Auli [↑](#footnote-ref-2)
3. *Speaker Recognition from Raw Waveform with SincNet*, by Mirco Ravanelli, Yoshua Bengio  [↑](#footnote-ref-3)
4. *Deep Neural Network Embeddings for Text-Independent Speaker Verification*, by David Snyder, Daniel Garcia-Romero, Daniel Povey, Sanjeev Khudanpur  [↑](#footnote-ref-4)