école normale supérieure paris-saclay

université PARIS-SACLAY

M1 E3A

TER - Travail d'Études et Réalisation

Reconnaissance vocale pour un laboratoire

Auteurs :
M. Etienne Stransky
M. Saša Radosavljevic

Encadrants :
M. Pascal Varoqui
Dr. Michel Lavner

Version du
30 décembre 2021

Table des matières

1 Introduction

Si la reconnaissance de la parole par les machine semble aujourd'hui être quelque chose d'acquis dans de nombreuses applications, cela a longtemps semblé n'être qu'un rêve inaccessible L'idée de commander une machine directement par la voix, plutôt que par le biais de boutons semblait une fantaisie de romans. Cependant, au cours du dernier siècle, les technologies se sont développées et, aujourd'hui, on retrouve de tels systèmes dans de nombreuses applications. Des lampes commandables par la voix aux assistants automatiques en passant par les outils d'aide aux personnes handicapées, la reconnaissance vocale est aujourd'hui une technologie déployée dans de très nombreux domaines.

Au cours des décennies qu'a pris le développement de ces systèmes, les techniques ont été amenées à évoluer. Si les premières machines n'étaient capables que de reconnaître un échantillon de voix à partir d'un ensemble défini, de nouvelles techniques ont émergé pour conduire, aujourd'hui, à des systèmes capables, en temps réel, de transcrire de façon complètement automatique un discours. Cela s'est fait grâce à deux progrès majeurs que sont le développement d'algorithmes de plus en plus fidèles et l'augmentation considérable de la puissance de calcul des ordinateurs, pour déployer des algorithmes de plus en plus sophistiqués. Le développement des réseaux de neurones à partir des années 2000 a également joué un rôle considérable pour améliorer les performances des systèmes de reconnaissances vocale et a permis à de nouveaux systèmes, plus rapides et plus fidèles de se déployer.

L'objectif de ce TER est de concevoir une application de reconnaissance vocale pour aider au comptage de cellules dans un laboratoire d'analyses médicales afin de pouvoir se dispenser d'utiliser des compteurs à boutons, simples mais peu pratiques. Le TER s'est donc concentré sur cet objectif avec plusieurs phases que sont la recherche bibliographique sur les techniques de reconnaissance vocale, le choix de l'outil de développement le plus adapté et enfin la réalisation d'une application de reconnaissance vocale satisfaisant le cahier des charges.

2 État de l'art : Principes de la reconnaissance automatique de la parole

La reconnaissance automatique de la parole a été développée à partir des années 1950 dans les laboratoires Bell, et n'a eu de cesse de s'améliorer au cours des années suivantes, et fait encore l'objet de recherches de nos jours. Le but de la reconnaissance automatique de la parole est d'identifier des mots qui ont été prononcés par une personne, afin de retranscrire ces mots sous une autre forme, le plus souvent du texte.

Lorsqu'une personne prononce un mot ou une phrase, elle génère un son qui peut être mesuré et analysé. Un mot est constitué d'unités sonores fondamentales appelées les phonèmes, dépendants de la langue, qui permettent de caractériser précisément la phonétique d'un mot.

Par exemple, le mot courant "Allô" se note phonétiquement $\backslash a.lo \backslash$, et possède donc les phonèmes a, l et o. La reconnaissance automatique de la parole peut donc se décomposer en deux parties : identifier les phonèmes qui ont été prononcés dans une phrase, puis effectuer une correspondance avec des lettres pour obtenir une transcription du texte. Nous nous focaliserons ici principalement sur la première partie de ce processus de reconnaissance.

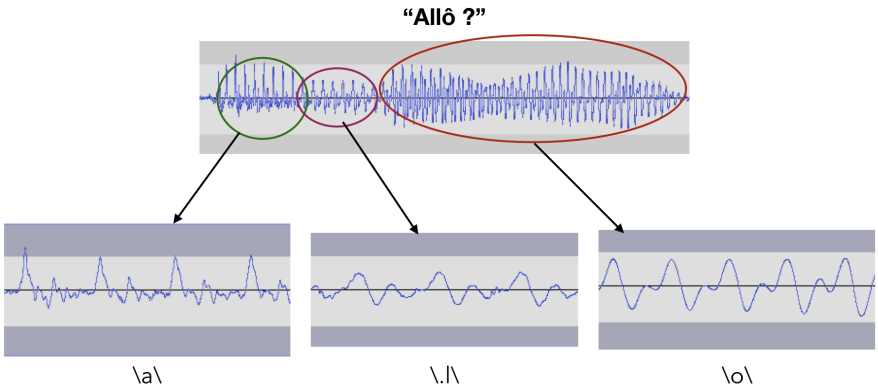


FIGURE 1 – Chronogramme du mot ”allô”

2.1 Les méthodes de caractérisation du son

La nécessité de caractériser les sons

Ainsi, la reconnaissance automatique de la parole se focalise sur la reconnaissance de phonèmes au sein d'un son capté. Cependant, pour que cette reconnaissance soit efficace, il faut caractériser précisément ces phonèmes pour pouvoir les reconnaître. En effet, il serait possible, pour reconnaître ces phonèmes, d'effectuer une simple comparaison entre le son reçu et des signaux de référence correspondant aux différents phonèmes (en calculant la corrélation entre les deux signaux par exemple).

Cette technique présente l'énorme inconvénient d'être particulièrement sensible aux variations du signal puisqu'une simple accélération (parler plus vite) peut fortement dégrader les performances d'une telle approche. De plus, les gens ayant tous des timbres de voix différents, ils produisent des sons à des fréquences différentes. Pour avoir une reconnaissance viable sur des échantillons sonores différents, il est donc nécessaire d'identifier des paramètres acoustiques dans le son, indépendants du locuteur ou des conditions précises de l'enregistrement, pour pouvoir effectuer des comparaisons restreintes et identifier uniquement l'information linguistique contenue dans le signal sonore. Il existe différentes approches, mais nous ne nous intéresserons qu'à deux d'entre elles.

L'analyse par prédiction linéaire

Pour caractériser un son, il est possible de faire appel à une méthode d'analyse par prédiction linéaire. Cette méthode consiste à faire l'hypothèse que le son capté n'est que le résultat du passage d'un signal d'excitation (impulsion ou bruit blanc) par un filtre linéaire. Sachant que, comme les systèmes de calcul sont numériques, ce filtre est donc de type AR (auto-récursif) ou ARMA (auto-récursif à moyenne ajustée). En partant du signal mesuré et en faisant une hypothèse sur la forme du signal d'excitation, nous pouvons donc estimer les coefficients de ce filtre via différentes méthodes.

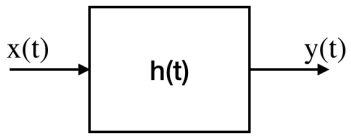


FIGURE 2 – Passage d'un signal x(t), impulsion ou bruit blanc, par un filtre

Pour permettre un bon fonctionnement de cette méthode, et pour que l'hypothèse du filtre linéaire soit valide, il faut effectuer la prédiction linéaire sur de courts échantillons du signal, comme pour l'analyse spectrale. Ainsi, on obtient, pour différentes sections du signal sonore, les coefficients du filtre linéaire, permettant de caractériser le son. En effet, la fréquence principale liée au système vocal du locuteur est facilement identifiable car la réponse fréquentielle du filtre présente un pic en cette fréquence, et la méthode n'est pas sensible aux variations du débit de parole.

L'identification par calcul des coefficients MFCC

Les MFCC (Mel-Frequency Ceptral Coefficients) sont des coefficients cepstraux calculés sur un signal pour pouvoir l'identifier. La technique d'identification par MFCC présente l'avantage d'être sensible à la même échelle que l'oreille humaine ; autrement dit, la différence entre deux signaux caractérisés par MFCC ressemblera à la différence de perception de ces deux signaux par l'oreille humaine ce qui, dans le domaine de la reconnaissance vocale, est d'un grand intérêt.

Le calcul des coefficients MFCC se fait de la manière suivante :

1. Le signal passe par un filtre passe-haut pour augmenter l'importance des hautes fréquences, et est fenêtré pour obtenir de courts morceaux du signal, qui se recouvrent (un même échantillon peut servir à calculer les coefficients de plusieurs segments).
2. On calcule la transformée de Fourier discrète du signal.
3. On passe l'amplitude de cette transformée de Fourier par un "filtre" de Mel. Celui-ci change l'échelle de fréquences du spectre du signal par la formule : $f' = xlog(1 + \frac{f}{y})$, où x et y sont des coefficients prédéfinis. Ainsi, les amplitudes du spectre restent inchangées, mais sont décalées sur l'échelle des fréquences.
4. On calcule le logarithme de ce spectre.
5. On calcule la transformée en cosinus discrète (équivalente à une transformée de Fourier) de ce spectre, ce qui donne les coefficients cepstraux associés à l'intervalle de temps étudié.

FIGURE 3 – Schéma de principe du calcul des coefficients MFCC

La technique d'identification par calcul des MFCC est très couramment utilisée dans de nombreux systèmes de reconnaissance vocale car, si elle est gourmande en ressources (il faut calculer deux transformées de Fourier), le fait que les coefficients reflètent la perception humaine permet d'amplifier les différences entre coefficients dans les bandes de fréquences couramment utilisées par l'Homme, et ainsi d'augmenter la perception du dispositif dans les fréquences utiles.

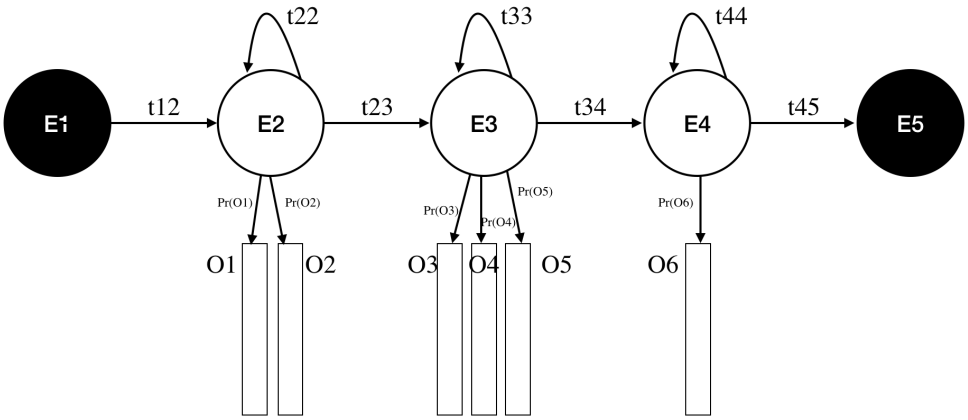


FIGURE 4 – Modèle markovien caché, utilisé en reconnaissance vocale

2.2 L'identification des phonèmes

À la suite de l'étape de caractérisation du son, nous obtenons des vecteurs contenant des valeurs caractérisant les différents segments de l'échantillon à analyser. L'étape suivante consiste à relier ces valeurs à des valeurs de référence pour pouvoir déduire le phonème qui a été prononcé. Pour cela, la technique la plus utilisée est l'utilisation d'un modèle de Markov caché. Il s'agit d'une machine à états probabiliste avec deux processus aléatoires indépendants : l'un donne les probabilités des différentes transitions à chaque état, l'autre la probabilité d'avoir différentes sorties pour chaque état.

Pour la reconnaissance automatique de la parole, le modèle de Markov caché est constitué de plusieurs états avec une transition unidirectionnelle (lorsqu'on quitte un état, on ne peut plus y revenir). Les états du début et de la fin correspondent à des silences, sans observations, tandis qu'aux autres états sont associées différentes observations, avec différentes probabilités de se produire, les observations correspondant aux caractéristiques des sons établies précédemment.

Ainsi, le modèle markovien est caractérisé par son nombre d'états, les probabilités des différentes transitions ainsi que les probabilités des différentes observations pour chaque état. Ce modèle donne une représentation complète d'un phonème, qui peut être amené à évoluer en fonction des phonèmes le précédant ou le suivant. En ayant des modèles markoviens pour tous les phonèmes d'une langue donnée, il est enfin possible d'estimer le phonème entendu grâce à la méthode du maximum de vraisemblance : le phonème entendu est celui qui maximise la vraisemblance des observations, autrement dit :

$$\hat{\phi} = arg\max_{\phi}[Pr(O/\phi)] \tag{1}$$

Où ϕ représente le modèle markovien associé au phonème et O le vecteur des observations. La recherche de ce minimum se fait par des méthodes numériques d'optimisation, de complexité variable.

Le principal problème de la reconnaissance de la parole est maintenant de trouver un modèle markovien correspondant à la langue à identifier. En effet, pour permettre une identification robuste, peu sensible aux changements de locuteurs, de ton ou de contexte, la construction de ce modèle devient très compliquée. La construction de ce modèle se fait donc souvent par apprentissage machine : on dispose d'une très grande base de données contenant tous les échantillons sonores nécessaires, et on construit un modèle markovien à partir des statistiques linguistiques sur la base de données. Cette partie d'apprentissage est généralement la partie la plus complexe, car c'est en fonction de l'adéquation entre le modèle markovien et les données à analyser que le système gagnera ou perdra en fidélité.

2.3 Modèles de langage : passer du phonème à la phrase

Une fois les phonèmes identifiés dans le signal sonore, le système doit encore parvenir à déterminer les lettres, puis les mots réellement prononcés. Pour cela, il peut s'appuyer sur un modèle lexical et un modèle de langage.

Un modèle lexical représente une liste de tous les mots possibles, avec sa prononciation. Ainsi, à une suite de phonèmes correspondra un mot, que le système peut identifier. Cette approche présente l'inconvénient de nécessiter un très grand travail, puisque tous les mots possiblement identifiables (y compris les noms propres) doivent y être recensés, et leur phonétique y être notée. Ce modèle peut cependant être construit de façon automatique, en utilisant les règles de prononciation de la langue (la lettre 'a' se prononce $\backslash a \backslash$, mais la combinaison 'ai' se prononce $\backslash e \ \backslash$, etc.), mais cela rend peu compte des exceptions et nécessite un paramétrage particulièrement efficace.

Un modèle de langage représente simplement les associations de mots les plus courantes dans une langue. Cela permet au système de reconnaissance de mieux identifier des mots qui forment un enchaînement logique, et ainsi de construire des phrases qui ont du sens. Par exemple, les mots 'pain' et 'bain' ayant une prononciation très similaire, le modèle de langage permet de trouver l'enchaînement de mots qui sera le plus commun dans la langue. Ainsi, la phrase 'Acheter du pain à la boulangerie' étant plus courante dans la langue française que 'Acheter du bain à la boulangerie', un modèle de langage permettra de trouver le mot le plus adapté. Cependant, si ce modèle améliore grandement la qualité de la transcription, il n'est pas indispensable pour permettre le fonctionnement de la plupart des systèmes de reconnaissance automatique de la parole.

2.4 Les nouvelles techniques : utilisation de réseaux de neurones

Depuis les années 2010, avec notamment les recherches de Baidu, de plus en plus de systèmes reposent désormais sur des réseaux de neurones récurrents qui permettent d'obtenir des résultats de façon plus rapide qu'avec un modèle de Markov, ce qui revêt d'un grand intérêt dans les systèmes embarqués et la reconnaissance en temps réel. C'est sur ces technologies que repose l'identification des sons de beaucoup de systèmes de reconnaissance modernes.

Principe d'un réseau de neurones artificiel

Les réseaux de neurones sont des systèmes complexes qui font l'objet de nombreuses recherches pour améliorer les systèmes existants. Dans le principe, le but est d'imiter artificiellement les réseaux de neurones des organismes vivants (notamment ceux présents dans nos cerveaux). Il y a trois couches dans un tel système, comme détaillé en Figure 5 : une couche d'entrée des données, une couche de sortie et une couche intermédiaire, comportant de nombreux neurones reliés entre eux. Ces connexions entre neurones sont ce qui permet de passer d'une valeur en entrée à une valeur de sortie du système. Le processus d'apprentissage de ces réseaux cherche à établir des liens entre les différents neurones et à fixer paramétrer ces liens. L'apprentissage consiste à envoyer des valeurs d'entrée, à mesurer les valeurs de sortie et à quantifier l'erreur qui existe. C'est en fonction de cette erreur que se réalise l'ajustement des différents paramètres du réseau de neurones.

Une fois l'entraînement réalisé, nous obtenons un réseau de neurones capable d'affecter de donner des valeurs de sortie en fonction de valeurs d'entrée. L'efficacité, la précision et l'adaptabilité du système dépendent notamment des échantillons entrés et des paramètres de l'entraînement.

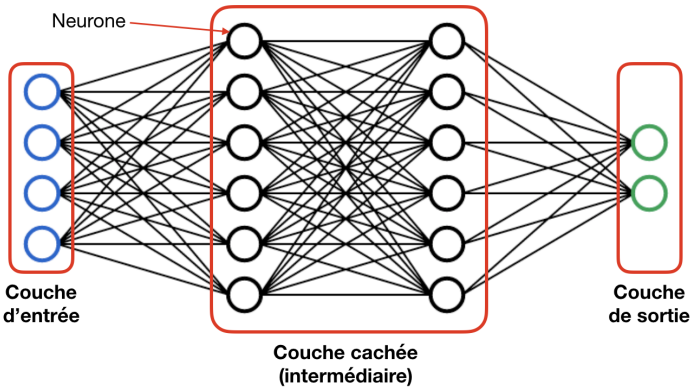


FIGURE 5 – Illustration du fonctionnement d'un réseau de neurones classique
Source : `https://victorzhou.com/series/neural-networks-from-scratch/`

Principe des réseaux de neurones récurrents (RNN)

Classiquement, un réseau de neurones utilise un ensemble de données, qui peuvent par exemple être figées dans le temps. Ainsi, l'enchaînement entre plusieurs entrées n'impacte pas le résultat final : peu importe l'ordre dans lequel les entrées se suivent, le résultat ne changera pas. Dans un système de réseau de neurones récurrent, au contraire, la succession entre les entrées a un impact sur le résultat en sortie. En effet, dans ce type de système, les échantillons sont enregistrés à la suite les uns des autres. Chaque échantillon est relié à un réseau de neurones. Ceux-ci se suivent, de sorte à ce que le résultat à l'instant t dépende des entrées à l'instant t, mais également aux instants précédents. La structure de ce type de réseaux est illustrée en Figure 6.

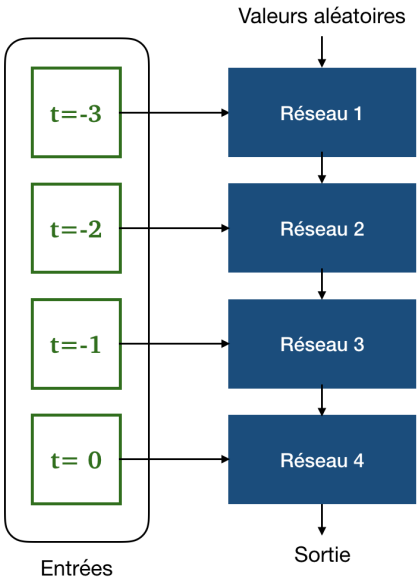


FIGURE 6 – Illustration de la structure d'un réseau de neurones récurrent

L'agencement entre ces réseaux donne une structure globale du système, qui peut varier. Il est également possible d'avoir plusieurs réseaux de neurones qui s'enchaînent à un même instant, pour obtenir de meilleurs résultats. Étant donné que, pour la reconnaissance vocale, c'est l'agencement entre les sons qui forme le caractère, les réseaux de neurones récurrents forment une bonne solution pour permettre une reconnaissance

adéquate.

Amélioration des technologies

Actuellement, les recherches portent sur l'amélioration des performances du système, en termes de vitesse et de fiabilité. L'utilisation de réseaux en Deep-Learning est de plus en plus courante, et les recherches portent principalement sur leur exploitation et la combinaison à des systèmes plus classiques pour permettre de meilleurs résultats. Un frein pour le développement de cette technologie est le nombre d'échantillons nécessaire pour obtenir une reconnaissance fidèle, ce qui donne un grand avantage aux grandes entreprises du numérique, qui ont accès à de grandes bases de données. Cette technologie fait donc encore l'objet de nombreuses recherches et est sujette à nombre d'innovations.

3 Consignes

Le TER a pour objectif de concevoir un système de reconnaissance automatique de la parole dans un cadre médical. En effet, lorsqu'un médecin analyse des échantillons de sang, il est nécessaire de compter les cellules présentes dans l'échantillon afin de les classer et d'arriver à des conclusions. Ainsi, le médecin observe l'échantillon à l'aide d'un microscope et doit noter les cellules présentes, ce qui l'oblige à lever les yeux de son observation, résultant en une légère perte de temps à chaque cellule comptée et une perte de repère d'observation. Le TER a pour fin de concevoir un système auquel il serait possible de dicter les cellules observées, sans avoir à quitter des yeux le système. Vu le contexte du système, un certain nombre de points doivent être respectés dans notre cas :

1. Le système doit pouvoir fonctionner entièrement sur une machine locale, sans aucune connexion Internet. Ce critère est impératif, vu le contexte médical de l'application, car une liaison extérieure pourrait présenter une faille de sécurité, et potentiellement permettre un accès aux données du laboratoire.
2. Le système doit fonctionner sur une machine de type Raspberry Pi, avec un code en langage Python. Cela permet une grande flexibilité, ainsi qu'une implémentation matérielle peu coûteuse, mais fiable.
3. Le système doit être multi-utilisateurs, puisque plusieurs personnes différentes peuvent être amenées à effectuer la procédure de comptage. Cela implique donc la nécessité d'une certaine robustesse aux changements de voix. Le système doit également s'accommoder du bruit qui règne dans les laboratoires d'analyses médicales.
4. Le système doit reconnaître des mots issus du domaine de la biologie. Comme ces mots ne sont pas courants, aucun système conventionnel de reconnaissance vocale n'est conçu pour les reconnaître directement. Ainsi, il semble indispensable qu'il soit possible d'ajouter des mots personnalisés à la bibliothèque courante des mots disponibles.
5. Le système doit être capable de reconnaissance temps réel et présenter, via une interface graphique, ses résultats à l'utilisateur pour qu'il puisse vérifier les résultats, et éventuellement corriger ses erreurs.

4 Les logiciels explorés pour effectuer la reconnaissance automatique de la parole

Cette section vise à présenter les différentes options de reconnaissance vocale que nous avons explorées avec les essais réalisés, les avantages et les inconvénients des différentes solutions que nous avons testées, et finalement le choix vers lequel nous nous sommes orientés.

4.1 Jasper

Développé par deux étudiants de l'université de Princeton, Jasper a été développé en Python afin de permettre l'utilisation de la reconnaissance vocale et de la synthèse vocale sur Raspberry Pi (Modèle B). Projet open source, il est plutôt distribué comme assistant vocal mais possède tout de même une API et supporte une grande variété de moteur Text To Speech (TTS) et de Speech To Text (STT) pouvant fonctionner en local.

FIGURE 7 – Logo de Jasper

L'installation se fait avec une image système pré-compilée à mettre sur une carte SD (à l'aide d'un logiciel disk imaging par exemple). L'utilisation de Jasper a rapidement été mise de côté de par le manque d'information sur l'apprentissage d'un modèle de reconnaissance vocal (adapté à notre utilisation). On peut cependant utiliser le modèle Sphinx en tant que STT. Une autre raison a été car il n'a pas encore été porté vers python 3 et que Python 2 n'est plus supporté depuis Janvier 2020.

4.2 Sopare

Sopare (pour SOund PAttern Recognition), est un système de reconnaissance vocale basé sur la similarité entre les données sonores enregistrées et les données correspondant à des mots pré-enregistrés. Ce système est simple par rapport à d'autres applications de reconnaissance vocale, puisqu'il cherche moins à transcrire les sons qu'à trouver le mot, parmi ceux qu'il a appris, qui correspond le mieux à ce qui a été reçu.

FIGURE 8 – Principe de la reconnaissance par Sopare

Cette approche pose néanmoins le problème d'être peu robuste aux applications multi-locuteurs, puisqu'il faut alors un entraînement pour chaque personne qui l'utilisera. Nous avons rejeté cet outil parce qu'il n'est pas compatible avec Python 3 et que les documentations sont relativement rares. De plus, le développement semble suspendu depuis 2018, ce qui peut mener à des problèmes de compatibilité avec les systèmes modernes. Ainsi, nous avons fini par rejeter cette solution, qui semble pourtant adaptée à notre problématique précise.

4.3 PocketSphinx

Le projet PocketSphinx est un projet de reconnaissance vocale basé sur le système CMU Sphinx, un projet pionnier dans le domaine de la reconnaissance vocale. Ce système présente les avantages d'être libre et conçu

pour une reconnaissance locale. Il possède également les interfaces nécessaires pour permettre l'entraînement du système de reconnaissance vocale, et ainsi autoriser l'ajout de mots personnalisés.

FIGURE 9 – Logo de CMU Sphinx

Le système PocketSphinx a été conçu dans le but précis de fonctionner pour des systèmes embarqués, et notamment sur des téléphones intelligents. Il est ainsi particulièrement adapté aux applications Androïd, même s'il est censé être compatible avec les ordinateurs classiques. Cependant, du fait de l'âge de l'outil, celui-ci présente des failles (notamment sur la taille des bases de données pour l'apprentissage), et une grande partie de son équipe de développement s'est orientée vers le développement d'un autre système : Vosk. De plus, ayant eu des problèmes de bibliothèque pour son installation, nous avons préféré nous orienter vers d'autres systèmes de reconnaissance vocale pour le TER.

4.4 Vosk

Vosk

Vosk est un système de reconnaissance vocale relativement récent (son introduction date de 2019), dont le but est de proposer un système de reconnaissance vocale hors-ligne qui soit fiable, mais également accessible à la plupart des systèmes embarqués. Il est interfaçable avec le langage Python, et propose des modèles neuronaux pré-entraînés dans plusieurs langues (et de plusieurs tailles selon les contraintes de la machine cible), afin de permettre la réalisation d'essais de vitesse et de fiabilité.

FIGURE 10 – Logo de Vosk

Essais

Nous avons réalisé des essais sur cette interface pour tester la qualité du décodage en temps réel. Dans un premier temps, nous avons cherché à dicter l'alphabet à deux modèles français : l'un simple (d'environ 300Mo), l'autre bien plus complexe (1,5Go). Sur le premier modèle, nous avons obtenu le résultat suivant : "a b c des peu f j'ai h y j car elle aimen au pays que 1'ere s t u v w x y z", ce qui semble peu fiable. L'autre essai nous a donné l'alphabet sous sa forme complète ("a b c d e f g h i j k l m n o p q r s t u v w x y z") mais, étant donné que ce modèle est bien plus complexe, il ne semble pas nécessairement adapté aux systèmes embarqués. De plus, aucun des deux modèles n'a su reconnaître les cellules à identifier.

Au vue des capacités de reconnaissance de l'outil avec les modèles pré-entraînés, la documentation et la capacité de reconnaissance temps réel, Vosk nous a semblé être un choix pertinent. Il faut cependant également une façon d'entraîner des modèles de langage appropriés à notre application. Dans le cas de Vosk, cela se fait grâce à l'outil de développement Kaldi.

Kaldi

Kaldi est un outil de développement destiné à permettre le support de la reconnaissance vocale pour d'autres logiciels. Il s'agit d'une initiative datant de 2009 afin de permettre un accès libre à des outils pour la reconnaissance vocale, qui soient de plus facilement modifiables. Il s'agit donc d'un outil de développement très complet, qui a pour vocation à permettre aux développeurs d'expérimenter et de choisir la technique de reconnaissance vocale qui leur convient le mieux.

FIGURE 11 – Logo de Kaldi

Cependant, s'il nous a été possible d'installer l'outil Kaldi, celui-ci est testé par le biais d'exemple, donc beaucoup sont mal documentés. Il est donc difficile de comprendre pourquoi certains ne fonctionnent pas. Mais, le principal problème est que Vosk n'est compatible qu'avec certains modèles développés grâce à Kaldi, et la documentation à ce sujet est difficile à trouver. Il ne fait aucun doute qu'il nous aurait été possible d'utiliser cet outil pour le TER mais, par souci d'économiser du temps, nous avons préféré abandonner cette solution.

4.5 Deepspeech

À propos

Deepspeech est un moteur STT (Speech To Text) open-source et embarqué (local, appareil électronique), développé par la fondation Mozilla, connue pour ses applications open-source destinées à permettre le développement des logiciels libres.

FIGURE 12 – Logo de Deepspeech

Le moteur Deepspeech est conçu pour pouvoir opérer en temps réel sur des cibles allant d'une Raspberry Pi 4 aux serveurs à hautes puissances de calcul (GPU). Ce moteur STT utilise un modèle de reconnaissance vocal entraîné à partir de Machine Learning basé sur les techniques du laboratoire en Intelligence Artificielle Baidu qui ne nécessite plus de dictionnaire de phonème ni du concept de phonème lui-même. Comme beaucoup de systèmes 'classiques', ce moteur caractérise le signal vocal à l'aide des coefficients MFCC mais, au lieu d'y associer un phonème pour ensuite en déduire une lettre ou un groupe de lettres, il utilise un système de réseau de neurones pour déterminer directement la lettre qui a été prononcée.

L'utilisation d'un réseau de neurones profond

L'utilisation d'un réseau de neurones récurrents (RNR) est une technique de reconnaissance vocale moderne, censée fonctionner dans des environnements plus bruyants que les autres moteur de STT mais également d'être plus robuste vis-à-vis de la réverbération et des variations entre les interlocuteurs. De façon générale, elle requiert également moins de calculs lors de la reconnaissance, ce qui la rend très adaptée aux systèmes embarqués. Elle présente néanmoins l'inconvénient de nécessiter un processus d'apprentissage plus long et plus coûteux en ressources. Elle rend également le système plus sensible aux erreurs durant l'apprentissage, car une erreur sur une lettre peut avoir des répercussions sur les autres, là où un modèle Markovien possède des systèmes séparés pour tous les phonèmes.

Essais préliminaires

Pour tester la reconnaissance de l'outil, nous avons utilisé un modèle de reconnaissance pré-entraîné. Malheureusement, il n'en n'existe pas en français, et nous avons donc utilisé le modèle anglais. En lui dictant l'alphabet (en anglais), nous avons obtenu le résultat suivant : "a b c d e f g h i j k l m n o p q r s t you t do wi said". Cela retranscrit les paroles de manière relativement fidèle, sauf pour les dernières lettres de l'alphabet, qui sont moins utilisées. Notons également que la lettre "w", étant prononcée "double u" en anglais, il semble normal que le système ait davantage de difficultés à la reconnaître. Nous avons également vérifié le bon fonctionnement du système en temps réel. L'absence d'un modèle français pré-entraîné est cependant un défaut pour notre application.

4.6 Le choix final de l'outil

Après différents tests sur les outils disponibles, notre choix pour le TER s'est finalement porté sur Deepspeech. Ce choix a été motivé par la disponibilité de documentations, notamment sur l'apprentissage des modèles.

L'installation sur Raspberry Pi a également été très bien guidée. Son utilisation nécessite alors des fichiers de modèles qui ont passés l'étape d'apprentissage (fichier .pbmm, .scorer) et permettant ainsi de transcrire des fichiers audio en texte. L'utilisation en temps réel à l'aide d'un microphone est réalisée avec l'ajout d'une fonction python mic_vad_streaming disponible dans les exemples de Deepspeech. C'est à partir de cette fonction que l'on va récupérer les résultats de la reconnaissance vocale.

De plus, l'outil ayant un développement récent, il est compatible avec Python 3, et il semble plausible qu'il soit encore maintenu pendant plusieurs années, avec des mises à jour probables, ce qui permettra à l'outil d'être encore fonctionnel pendant plusieurs années.

5 Réalisation de l'outil de reconnaissance vocale

Dans cette partie, nous allons décrire les différentes étapes de la réalisation de l'outil de reconnaissance vocale pour répondre au problème du TER. Il s'agit de la mise en place de l'environnement d'entraînement, du processus d'entraînement, du paramétrage de la carte Raspberry Pi ainsi que du développement du logiciel final de reconnaissance.

5.1 Mise en place et entraînement du réseau de neurones

Cette partie traite du processus conduisant à l'obtention d'un modèle neuronal utilisable pour l'application de reconnaissance automatique de la parole.

Environnement de travail

Pour fonctionner convenablement et reconnaître la parole, DeepSpeech utilise un réseau de neurones qui est obtenu après un entraînement. L'entraînement est réalisé de façon autonome en utilisant l'outil *Tensorflow*. Cet entraînement nécessite de nombreux calculs, car il s'agit d'un processus itératif sur les échantillons pour obtenir le réseau de neurones conduisant à l'erreur la plus faible. C'est pourquoi une machine de calcul a été mise à notre disposition, sur laquelle il nous a été possible de nous connecter à distance pour réaliser les entraînements. Cependant, comme les entraînements ont nécessité l'installation de nombreux logiciels et la modification de certains paramètres de la machine, les droits du super-utilisateur de la machine étaient requis. Ainsi, c'est par le biais d'une machine virtuelle (sur laquelle nous avons tous les droits) que les entraînements ont pu être faits.

L'utilisation d'une machine virtuelle pose cependant le problème de l'optimisation du temps de calcul : l'entraînement de réseaux de neurones est généralement plus rapide s'il est réalisé sur la carte graphique de l'ordinateur. Or, l'utilisateur d'une machine virtuelle n'a pas un accès direct à la carte graphique : nous avons donc réalisé tous les calculs sur le processeur, et non la carte graphique. Finalement, cela n'a eu qu'une faible incidence sur le déroulement de notre TER, car les calculs se sont effectués rapidement même sur le processeur, mais il peut s'agir d'une modification à effectuer pour améliorer les performances.

Acquisition et préparation des échantillons de voix

Pour réaliser l'entraînement nécessaire à la conception du modèle neuronal, nous avions besoin d'échantillons de voix avec la transcription associée. Ainsi, c'est en comparant les transcriptions issues du modèle avec les transcriptions réelles que le système d'entraînement est capable de déterminer les paramètres à ajuster, et ainsi fabriquer un réseau de neurones capable d'une reconnaissance fidèle. Le processus complet conduisant à l'obtention des fichiers nécessaires à la reconnaissance est décrit en Figure 13.

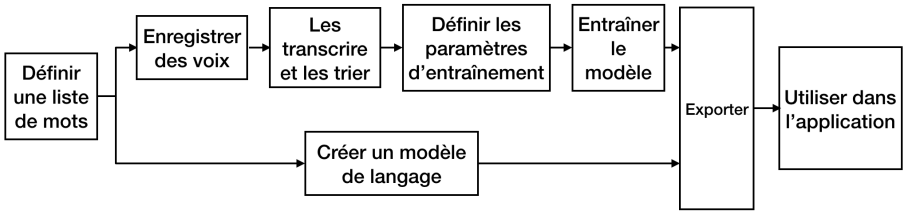


FIGURE 13 – Résumé du processus permettant l'entraînement du réseau de neurones pour la reconnaissance

Vous pourrez également retrouver une acquisition de données embarquée sur STM32F746G réalisée pour le mini-projet d'informatique industrielle.

Tout d'abord, nous avons dû mettre en place une liste de mots appelée Dictionnaire de vocabulaire sur laquelle notre système de reconnaissance vocale va se baser. Cette liste est composée des 28 mots suivants :

1. Start
2. Stop
3. Erreur
4. Neutro
5. Neutrophile
6. Éosino
7. Éosinophile
8. Baso

9. Basophile
10. Lympho
11. Lymphocyte
12. Mono
13. Monocyte
14. Blaste
15. Tricho
16. Tricholeucocyte
17. Sézary
18. Autre
19. Un
20. Deux
21. Trois
22. Quatre
23. Cinq
24. Six
25. Sept
26. Huit
27. Neuf
28. Dix

Cette liste est exhaustive mais non définitive (les noms des cellules que l'on peut retrouver pour cette application ainsi que des mots utiles au fonctionnement de l'application sont listés ici). Le postulat de base est alors que plus l'on possède de données (personnes différentes) plus notre modèle sera précis. En effet, la validation croisée lors de l'apprentissage permettra d'obtenir la meilleure répartition entre données d'apprentissage et données de validation. Nous avons alors développé un script Matlab permettant d'enregistrer des personnes volontaires et de séparer automatiquement les voix en différents fichiers audio (Voir annexe).

L'algorithme est assez simple et est le suivant :

```
Définition du dictionnaire
Définition de la phonétique associée
Définition de la fréquence d'échantillonnage (16kHz)

Requête du nom de l'utilisateur

Pour tous les mots du vocabulaire :
    Afficher le mots à prononcer et sa phonétique si existante
    Attente d'appui de l'utilisateur pour l'enregistrement
    ...Enregistrement d'un temps proportionnel à la longueur du mot
    Enregistrement dans un .vaw
    Demande de validité ou de répétition à l'utilisateur
fin
```

Une fois les données collectées vient une nouvelle étape : il faut les rendre lisibles par le système d'entraînement de DeepSpeech. Celui-ci requiert d'avoir trois fichiers *.csv*, correspondant aux phases *train* (entraînement), *dev* (évaluation pendant l'entraînement) et *test* (évaluation finale). Ces fichiers doivent contenir l'intégralité des fichiers sonores utilisés pour l'entraînement, leur taille ainsi que leur transcription. Pour permettre la création automatique de ces fichiers à partir des échantillons récupérés, nous avons écrit un script en Python qui génère les fichiers csv et répartit les fichiers sonores associés dans trois dossiers. L'algorithme utilisé effectue les opérations suivantes :

1. Un répertoire 'entree' contient, séparés dans des dossiers par personne, tous les échantillons sonores récoltés avec un nom du type : "nom_de_la_personne_mot.wav"
2. On parcourt tous les fichiers du répertoire "entree" et on lit les chemins de tous les fichiers présents
3. On trie ces fichiers en fonction du nom prononcé (celui-ci est contenu dans le nom du fichier)
4. Pour chaque mot on répartit, de façon aléatoire les fichiers associés en trois catégories : train, dev et test, de sorte à avoir 60% de fichers train, 30% de dev et 10% de train.
5. On crée un fichier .csv qui, pour chaque catégorie, contient le nom du fichier, le nom prononcé (tiré du nom) ainsi que sa longueur (en nombre d'échantillons)
6. On copie les fichiers sonores dans les répertoires de sortie adéquats

À l'issue de ce processus de tri, nous obtenons un fichier contenant toutes les données triées, disponibles pour l'entraînement. Ainsi, avec ce procédé, en faisant en sorte que les fichiers portent des noms adéquats, il est possible d'acquérir un nombre important de données pour permettre l'entraînement du système.

Paramètres de l'entraînement

Une fois le corpus constitué et les fichiers préparés, étant donné que nous disposons d'un environnement équipé de tous les modules nécessaires à l'entraînement du réseau de neurones, nous pouvons enfin lancer l'entraînement. Pour cela, nous lançons l'environnement virtuel d'entraînement puis le programme Python *Deepspeech.py* qui réalise automatiquement tout l'entraînement.

```
1 $ python -u DeepSpeech.py --#Tous les paramètres
```

Il faut cependant lui préciser certains paramètres pour permettre un entraînement dans de bonnes conditions. Nous devons donc préciser les paramètres suivants :

— Les chemins vers les fichiers csv reliant les données des phases train, dev et test de l'entraînement.

— Le nombre de fichiers à propager à travers le réseau à chaque itération. Ce paramètre donne le nombre d'échantillons qui seront traités en même temps par le réseau de neurones. Plus ce nombre est important, moins il faudra d'itérations pour traiter l'intégralité des fichiers, mais cela augmente également la mémoire nécessaire pour réaliser le traitement. En règle générale, augmenter ce nombre (appelé également *batch size*) tend à augmenter la vitesse de l'entraînement, aux dépends de la précision.
Il n'existe cependant pas de règle universelle pour fixer ces tailles de façon optimale, et cela fait encore l'objet de recherches. Dans notre cas, nous utilisons des tailles relativement faibles (autour de 16 fichiers) car nous ne disposons que d'un nombre d'échantillons limité. Les meilleurs résultats ont été obtenus pour une série utilisant 16 fichiers à l'entraînement, et 8 pour les étapes *dev* et *test*

— Le paramètre *n_hidden* qui précise le nombre de couches du réseau de neurones entre la couche d'entrée et la couche de sortie. Pour des modèles complexes, ce nombre de couches doit être important pour permettre une reconnaissance la plus précise possible. Cependant comme, dans notre cas, nous utilisons un modèle simple, ce paramètre doit être réduit pour simplifier le modèle.

— Le paramètre *learning_rate* est un paramètre-clé de l'optimisation du modèle. Il s'agit du pas dans l'optimisation du réseau de neurones. Plus ce pas est grand, plus l'optimisation (et donc le processus d'apprentissage) sera rapide, mais cela se fait au risque de louper le minimum de l'erreur et donc de perdre en qualité. À l'inverse, un pas petit diminue ce risque, mais augmente également le temps de descente. Il faut donc le choisir avec soin pour obtenir des résultats précis.

— Le paramètre *reduce_lr_on_plateau* permet de modifier de façon dynamique le *learning rate* en le réduisant s'il y a un plateau lors de l'entraînement, c'est-à-dire après un certain nombre d'itérations durant lesquelles l'erreur ne s'est pas améliorée. Ainsi, cela permet d'avoir un taux d'apprentissage élevé au début, puis de le réduire à mesure que l'entraînement s'affine.

— le paramètre *dropout_rate* précise, à chaque itération, le pourcentage de neurones tués aléatoirement. Cela permet d'éviter qu'un modèle ne soit trop adapté à un ensemble d'entraînement précis. En tuant certains neurones, on crée des perturbations dans le réseau, comme ce serait le cas avec des données plus variées. Si le modèle résiste bien à ces perturbations, cela signifie qu'il est robuste, et donc plus adapté qu'un modèle présentant une erreur plus faible à l'origine, mais qui se dérgade à la moindre perturbation.

— Enfin, il faut également préciser le nombre d'itérations sur l'entraînement (le nombre d'*epoch*), c'est-à-dire le moment où celui-ci s'arrête. Il y a également des conditions d'arrêt, notamment sur l'erreur : si l'erreur ne descend plus au bout d'un certain nombre d'itérations, on considère qu'il faut arrêter. Il est possible de régler ce paramètre pour que l'apprentissage s'arrête automatiquement si l'erreur ne descend plus au bout d'un certain nombre d'itérations, ce qui signifie que l'entraînement a convergé. Cela est illustré par la Figure 14, qui montre l'évolution de l'erreur de prédiction en fonction du nombre d'entraînements sur une de nos séquences. Elle illustre bien l'utilité d'un arrêt prématuré de l'entraînement et l'existence d'un nombre optimal d'itérations à réaliser sur l'entraînement.

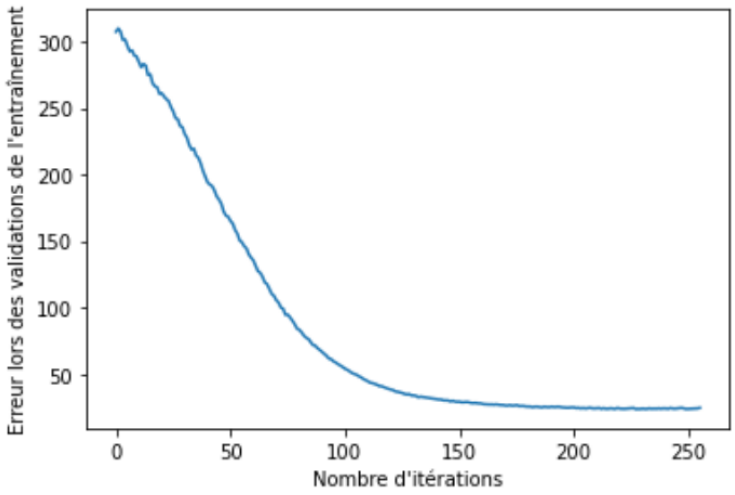


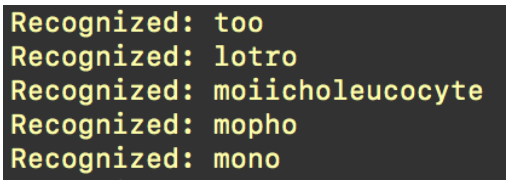
FIGURE 14 – Evolution de l'erreur en fonction du nombre d'entraînements

Une fois la commande lancée et l'entraînement terminé (dans notre cas, cela a pris généralement une vingtaine de minutes), nous pouvons récupérer le modèle du réseau de neurones. Celui-ci peut se présenter sous deux formes (paramétrables lorsqu'on lance l'entraînement) : soit un fichier .pb convertissable en .pbmm, conçu pour la version classique de Tensorflow (celle qui tourne sur des PC ou des serveurs), soit un fichier .tflite fonctionnant avec la version allégée de Tensorflow, destinée aux applications embarquées. Étant donné que le programme final doit fonctionner sur une carte Raspberry Pi, nous utilisons la version tflite. Cela a pour conséquence principale que, comme c'est une version allégée de tensorflow qui est utilisée pour la reconnaissance, la précision du modèle se retrouve nécessairement dégradée par rapport à une forme classique, conduisant à une dégradation des performances.

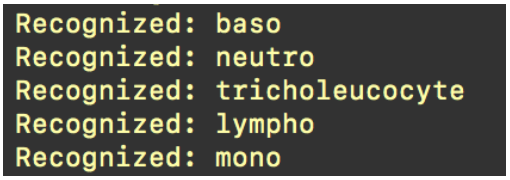
Création d'un modèle de langage

Une fois le réseau de reconnaissance vocale obtenu, celui-ci peut être utilisé tel quel pour reconnaître la parole. Cependant, son but est de reconnaître des lettres à partir de sons, c'est-à-dire que, à partir des mots et des sons qui lui ont été fournis, le système va identifier la séquence de lettres qui correspond, selon lui, le mieux au son qui a été entendu. Mais, comme chaque son est différent, la séquence de lettres reconnue peut ne correspondre à aucun mot. Pour forcer la reconnaissance de mots qui existent réellement, nous pouvons utiliser ce qui s'appelle un modèle de langue, ou encore un *scorer*. Celui-ci permet significativement d'améliorer la qualité de la transcription, comme l'illustre la Figure 15.

Pour construire le modèle de langage, DeepSpeech utilise l'outil Kenlm, un outil open-source utilisé par de nombreux outils de reconnaissance vocale. Dans notre cas, comme nous cherchons à identifier des mots isolés, nous lui mettons en paramètre la liste des mots à identifier par le logiciel. Pour plus de précision, DeepSpeech offre la possibilité de calculer des paramètres supplémentaires sur le réseau de neurones entraîné pour adapter le modèle de langage au réseau.



(a) Résultat d'une reconnaissance sans fichier scorer



(b) Résultat d'une reconnaissance avec fichier scorer

FIGURE 15 – Comparaison des résultats d'une reconnaissance vocale avec et sans fichier scorer
Mots prononcés : "baso, neutro, tricholeucocyte, lympho, mono"

Après exécution de l'outil Kenlm, nous obtenons le modèle de langage qui indique au système de reconnaissance quels sont les mots qui peuvent être attendus : ainsi, si un mot proche est reconnu ("sono" au lieu de "mono" par exemple), le résultat sera corrigé et, au contraire, si un mot qui n'a rien à voir avec la liste est entendu, il sera ignoré. Cela agit donc également comme une sécurité contre le bruit de fond car des mots prononcés qui ne rentreraient pas dans le cadre de l'expérience peuvent ainsi être ignorés.

Une fois le fichier *scorer* fabriqué, nous pouvons le télécharger ainsi que le réseau de neurones de l'ordinateur de calculs sur la carte Raspberry Pi, et ainsi commencer la reconnaissance vocale proprement dite.

6 Intégration de la reconnaissance vocale sur Raspberry

Tout d'abord, malgré la possibilité d'utiliser DeepSpeech sur des systèmes embarqués, il nécessite tout de même une certaine puissance de calculs afin notamment de réduire le temps de reconnaissance. Le choix du modèle de Raspberry s'est rapidement tourné vers le modèle 4 possédant 4 Go de RAM.

6.1 Installation de DeepSpeech

Pour installer et commencer à utiliser DeepSpeech, nous avons besoin d'une version Python 3.5 ou plus ainsi que du paquet d'installation pip3 associé. Il faudra par la suite porter une attention particulière à ne pas oublier le "3" dans les commandes lors de l'installation et de l'utilisation.

Avant d'installer DeepSpeech il faut créer un environnement virtuel,

```
$ virtualenv -p python3 ~/../deepspeech-venv/
```

En l'activant avec la commande suivante,

```
$ source ~/../deepspeech-venv/bin/activate
```

Nous utilisons maintenant le paquet pip3 pour l'installation de DeepSpeech. Il peut être installé avec la version utilisant un processeur graphique (deepspeech-gpu) mais n'en possédant pas sur la Raspberry, la commande utile est alors,

```
$ pip3 install deepspeech
```

Nous sommes fin prêts à utiliser les possibilités offerte par DeepSpeech pour la reconnaissance vocale. La commande deepspeech comporte un certain nombre d'arguments :

```
--model MODEL         Path to the model (protocol buffer binary file)
--scorer SCORER       Path to the external scorer file
--audio AUDIO         Path to the audio file to run (WAV format)
--beam_width BEAM_WIDTH
                        Beam width for the CTC decoder
--lm_alpha LM_ALPHA   Language model weight (lm_alpha). If not specified,
                        use default from the scorer package.
--lm_beta LM_BETA     Word insertion bonus (lm_beta). If not specified, use
                        default from the scorer package.
--version             Print version and exits
--extended            Output string from extended metadata
--json                Output json from metadata with timestamp of each word
--candidate_transcripts CANDIDATE_TRANSCRIPTS
                        Number of candidate transcripts to include in JSON
                        output
```

Les paramètres utiles pour notre utilisation sont donc le modèle (–model) qui est le fichier généré par l'apprentissage. Il peut avoir plusieurs terminologies, .pbmm ou .tflite. Ce dernier vient de Tensorflow Lite, ce type de fichier est optimisé pour le stockage et les systèmes embarqués (il ne peut pas être réentrainé par dessus). L'autre paramètres important est comme nous l'avons vu précédemment, le fichier fichier scorer qui est donné via –scorer.

Enfin, un programme supplémentaire est nécessaire pour notre utilisation, celui qui permet au modèle d'utiliser le microphone de la Raspberry en temps réel, appelé mic_vad_streaming disponible dans les exemples d'applications partagés sur Github

Une fois téléchargé, l'installation se fait alors avec le le fichier des plugins `requirements.txt`

```
pip3 install -r requirements.txt
```

On lance alors le programme python de la manière suivante,

```
python3 mic_vad_streaming.py -v (aggressiveness 0-3) -m (model tflite directory) -s
↪  (model scorer directory) -f (lecture depuis un .wav)
```

Nous verrons ce qu'apporte le paramètre d'agressivité sur le filtrage de la parole dans les résultats des différents modèles créés.

6.2 Modification du programme mic_vad_streaming.py

L'information principale à récupérer du programme mic_vad_streaming.py est le mot reconnu par notre modèle. Cette information se trouve à la ligne 194 du programme d'origine. Il a dont fallu rajouter une interface graphique pouvant fonctionner en parallèle. Le programme intégrant déjà les bibliothèques threading,

un premier essai a été d'utiliser Tkinter. Malheureusement celui-ci n'est pas thread-proof et possède une fonction bloquante malgré l'utilisation de threads.

Pygame

Nous avons alors utilisé l'interface graphique qui fonctionne très bien en multi-threading et possédant une très bonne documentation. Le seul point négatif étant l'affichage de texte sous forme d'image et donc non sélectionnable.

L'interface graphique ressemble donc à cela :

FIGURE 16 – Interface graphique de la reconnaissance vocale

L'utilisation est simple, la prononciation du mot "Start" permet de lancer le compteur, le mot "Stop" l'arrête. Lors de la prononciation d'un chiffre suivit du nom de la cellule, celle-ci est incrémenté du chiffre indiqué. La prononciation du mot "Erreur" annule la dernière action. Enfin, deux boutons sont présents. Le premier permet d'exporter le compteur sous format .csv et le second permet d'arrêter le comptage manuellement (on ne sait jamais ce qu'il peut se passer avec le micro).

7 Résultats

Après plusieurs entraînements de modèles, de tests, de vérifications et enfin de validation, nous avons à partir d'une trentaine de voix pu obtenir les résultats suivants. Pour tous les modèles et pour différentes valeurs d'agressivité, nous avons prononcé les mots 10 fois de manière naturelle qui puisse varier légèrement mais avec la même personne. Tous les essais ont été réalisés sur la carte Raspberry avec l'utilisation du fichier ".tflite".

7.1 Premier modèle avec des résultats significatifs

Pour ce modèle, tous les échantillons ont été utilisés pour l'entraînement. Nous verrons plus tard les raisons pour lesquelles certains mots étaient assez mauvais. De plus, tous les mots du corpus, abréviations et chiffres ont été utilisés.

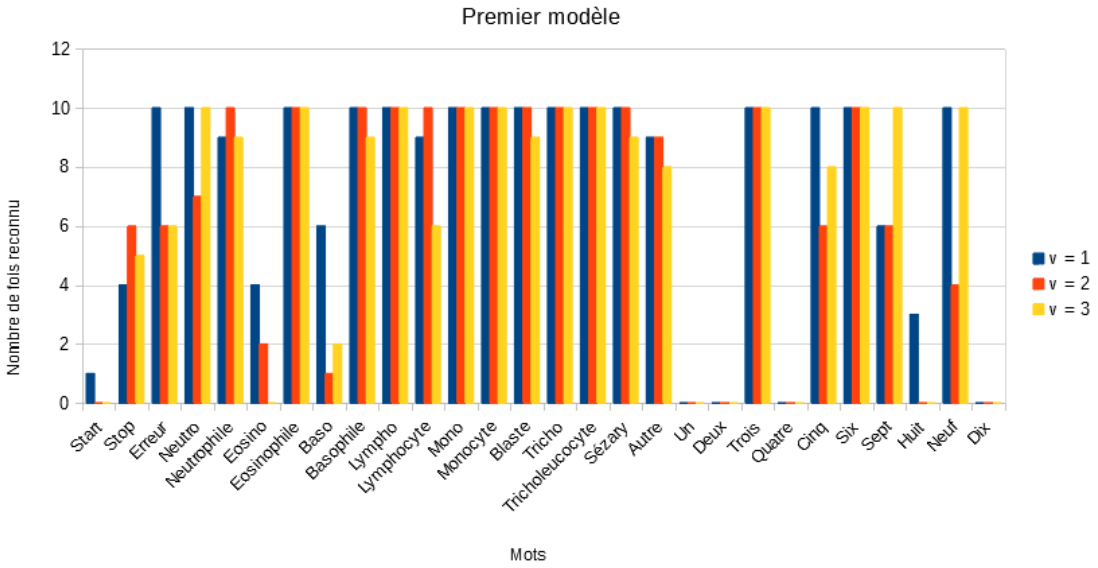


FIGURE 17 – Premier modèle valide

On remarque que certains mots ne sont pas du tout reconnus et que les abréviations posaient quelques soucis. Notre première hypothèse était alors que les abréviations, assez courtes, gênaient la reconnaissance des mots courts, sûrement du point de vue du faible nombre d'échantillons vocaux. Deuxièmement, nous pouvons choisir d'enlever le chiffre 1 qui est implicitement indiqué avec le nom et le nombre 10 qui ne semblerait jamais pouvoir être reconnu. Nous obtenons alors les fiabilités suivantes :

	v = 1	v = 2	v = 3
% Fiabilité	71,78	63,21	64,64
% Fiabilité sans 1 et 10	77,31	68,08	69,61

Ces résultats montre que les noms des cellules sont bien reconnues mais qu'il y a quelques soucis et que la fiabilité n'est pas assurée.

7.2 Modèle sans abréviations

Les noms des cellules abrégés n'étant pas fondamentalement nécessaire au bon fonctionnement du modèle, nous avons entraîné un modèle sans.

Cette fois-ci, on remarque des résultats très encourageants quant aux mots qui ne sont pas des nombres avec les fiabilités suivantes :

	v = 1	v = 2	v = 3
% Fiabilité	85,9	85,00	81,36
% Fiabilité sans 1 et 10	92	92	89

Résultat étonnant, les mots start, stop et erreur sont très bien reconnus cette fois-ci. Nous nous sommes alors posé la question quant à la validité des échantillons proposés. En effet, malgré la proposition de correction dans le programme Matlab, certains échantillons du mot "start" étaient vide, certaines fois des micro qui

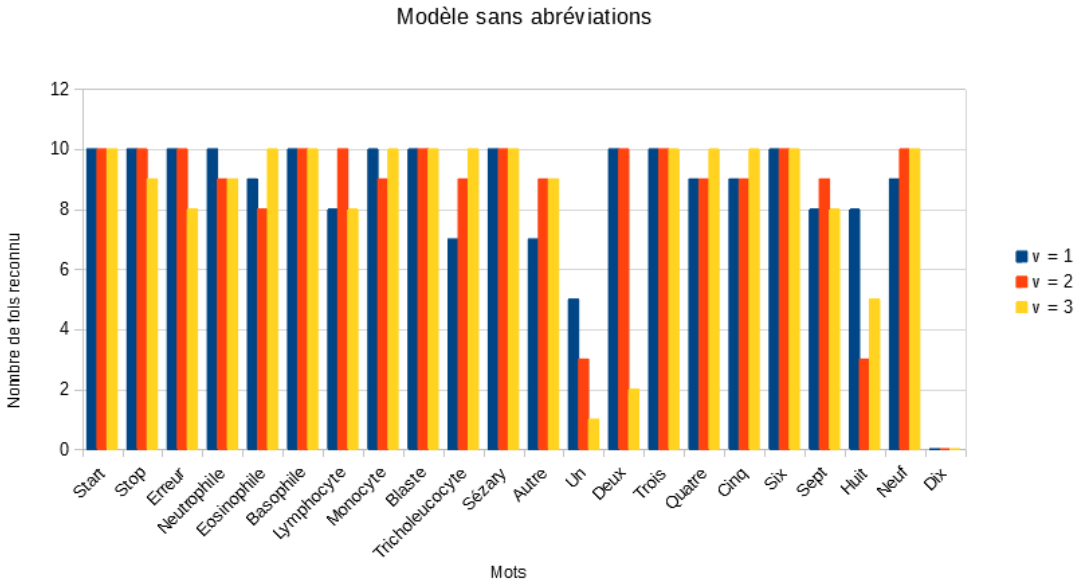


FIGURE 18 – Modèle sans abréviations

ne s'activent pas correctement, d'autres saturés et dans de rares cas des voix en fond qui se superposent. Rappelons également que le paramètre **v** correspond au filtrage du bruit, plus il est important et plus la voix est filtrée.

7.3 Modèle sans abréviations et base de données corrigée

Une fois tous les échantillons passés en revu, une petite proportion a été supprimée avant l'entraînement, nous donnant des résultats assez significatifs.

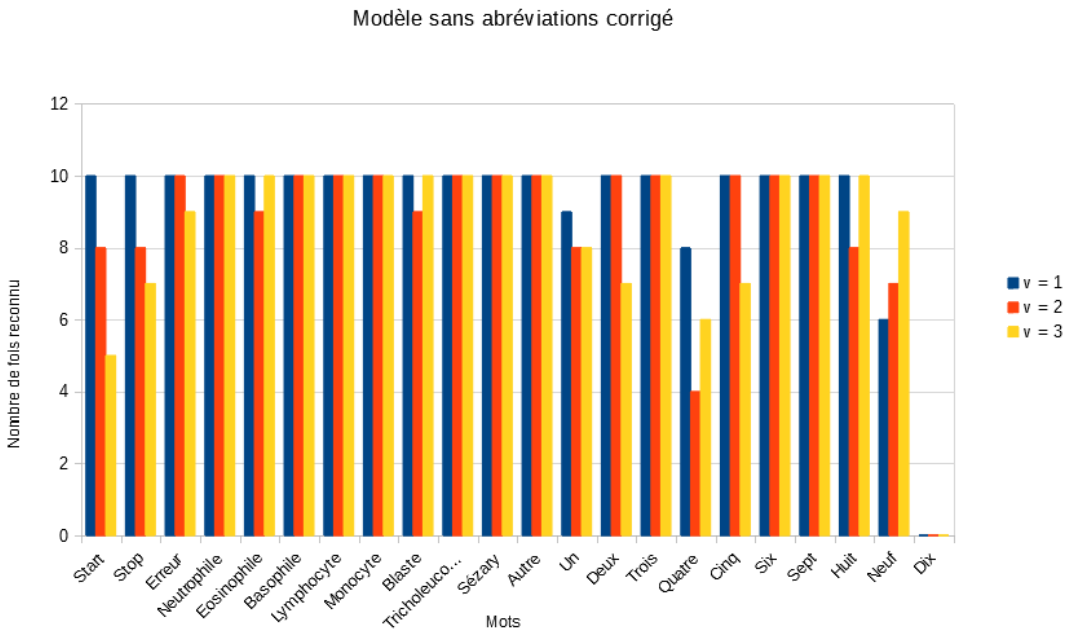


FIGURE 19 – Modèle sans abréviations et données corrigées

Ce modèle est alors le plus réussi avec des fiabilités supérieures à 90%

	v = 1	v = 2	v = 3
% Fiabilité	92,27	86,82	85,45
% Fiabilité sans 1 et 10	95	91,5	90

Un point important à souligner est le meilleur résultat pour une agressivité de 1 qui vient probablement du choix réalisé en amont quant au bruit. Effectivement, nous avons décidés de laisser un certain bruit être présent sur les échantillons, en partant du principe que l'environnement dans lequel l'utilisation sera faite était assez bruyant (visite du labo), évidemment un bon microphone et un peu moins de bruit permet tout de même d'avoir de meilleurs résultats.

Conclusion

Après une année scolaire complète passée à travailler sur ce projet de reconnaissance automatique de la parole, nous sommes finalement parvenus à obtenir un résultat fonctionnel pour l'application désirée. Pour cela, la première étape a consisté à analyser le sujet de l'étude pour comprendre les attentes. Ensuite est venu une grande étape de recherches bibliographiques, d'analyses et d'essais sur différents systèmes de reconnaissance vocale afin de trouver celui qui nous conviendrait le mieux. Une fois notre choix arrêté est venue l'étape de la réalisation avec, en parallèle, l'entraînement du réseau de neurones et la programmation du logiciel. Ce n'est qu'en ayant réalisé toutes ces étapes que nous avons pu finaliser ce TER.

Au cours de ce projet, nous avons acquis de nombreuses compétences, que ce soit en programmation, en informatique ou sur l'organisation et la planification d'un projet à long terme. Nous avons également étoffé nos connaissances, notamment dans le domaine du traitement du signal et de l'intelligence artificielle. Ce TER aura donc été pour nous une expérience très enrichissante à de nombreux égards, qui nous offre une première expérience en vue de pouvoir travailler dans le domaine de la recherche.

Bibliographie

Références concernant la théorie

[1] **Description de l'état de la reconnaissance vocale**

[1] Mohamed Bouaziz. *Voix & IA / Reconnaissance Automatique de la Parole.* https ://www.aquiladata.fr/voix-ia-reconnaissance-automatique-de-la-parole/. Mai 2020.

[6] **Description des systèmes de reconnaissance vocale classiques**

[6] Othman Lachhab. *Reconnaissance Statistique de la Parole Continue pour voix Laryngée et Alaryngée.* `https://hal.inria.fr/tel-01563766/document`. Juil. 2017.

[7] **Principe du fonctionnement d'un réseau de neurones récursif**

[7] *Les réseaux de neurones récurrents : des RNN simples aux LSTM.* `https://blog.octo.com/les-reseaux-de-neurones-recurrents-des-rnn-simples-aux-lstm/`. Oct. 2019.

[8] **Comparaison des performances RNN contre HMM**

[8] Kofi Appiah Mohammed Kyari Mustafa Tony Allen. *A comparative review of dynamic neural networks and hidden Markov model methods for mobile on-device speech recognition.* https ://link.springer.com/article/10.1007/s00521-017-3028-2. Juin 2017.

[9] **Explication du calcul des coefficients MFCC**

[9] Pratheeksha Nair. *The dummy's guide to MFCC.* https ://iopscience.iop.org/article/10.1088/1742-6596/1090/1/012046/pdf. Juil. 2018.

[11] **Utilisation du système LPC pour la reconnaissance vocale**

[11] W.S. Mada Sanjaya. *Speech Recognition using Linear Predictive Coding (LPC) and AdaptiveNeuro-Fuzzy (ANFIS) to Control 5 DoF Arm Robot.* https ://iopscience.iop.org/article/10.1088/1742-6596/1090/1/012046/pdf. 2018.

[13] **Explications simple du fonctionnement des réseaux de neurones**

[13] *The Principles Of Neural Network Functions Simplified.* `https://www.nixsolutions.com/blog/principles-convolutional-neural-networks-simple-words/`. Sept. 2018.

[15] **Description des systèmes de reconnaissance vocale**

[15] Khaled Zaabi. *Implémentation d'une méthode de reconnaissance de la parole sur le processeur de traitement numérique du signal.* `https://espace.etsmtl.ca/id/eprint/687/1/ZAABI_Khaled.pdf`. Juin 2004.

Références pour les différents systèmes de reconnaissance vocale

[5] **Description du système Kaldi**

[5] *Kaldi*. http://kaldi-asr.org/doc/.

[10] **Description du système PocketSphinx**

[10] *Open source speech recognition toolkit*. https://cmusphinx.github.io/. Oct. 2019.

[12] **Description du système Sopare**

[12] *Smarthome, speech recognition, voice control, Raspberry Pi and more*. http://www.bishoph.org/. Mar. 2018.

[14] **Description du système Vosk**

[14] *Vosk documentation*. https://alphacephei.com/vosk/. Mar. 2021.

Références pour les outils pratiques servant à la réalisation du TER

[2] **Informations sur les paramètres de l'entraînement de modèle de Deepspeech**

[2] Nicolas Can. *Mise en place de l'autotranscription*. https://www.esup-portail.org/wiki/display/ES/Mise+en+place+de+l%27autotranscription. Fév. 2021.

[3] **Guide de développement du modèle neuronal de Deepspeech**

[3] *DeepSpeech Playbook*. https://mozilla.github.io/deepspeech-playbook/. Fév. 2021.

[4] **Paramétrisation de l'entraînement du modèle Deepspeech**

[4] Vincent Foucault. *TUTORIAL : How I trained a specific french model to control my robot*. https://www.esup-portail.org/wiki/display/ES/Mise+en+place+de+l%27autotranscription. Déc. 2017.