ENSTA BRETAGNE

Projet de fin d'études

Mise en oeuvre de techniques d'intelligence artificielle pour la reconnaissance vocale

Cyril COTSAFTIS
CI 2020 - ROB

Tuteur école :
Luc JAULIN

Encadrant principal :
Marc MICHAU

Juillet - Décembre 2020

Remerciements

Je tiens à remercier tous les membres de l'équipe MMP pour leur accueil et leur bienveillance à mon égard. En particulier mes encadrants, Marc MICHAU et François CAPMAN, pour leur conseils avisés et la confiance qu'ils m'ont accordée. Je remercie également Margaux ALEXANDRIAN pour sa bonne humeur au quotidien.

Résumé

L'objectif de mon stage est de mettre en oeuvre des techniques d'intelligence artificielle pour la reconnaissance vocale. Pour cela, j'ai dû me familiariser avec les concepts fondateurs de la reconnaissance vocale (traitement du signal de la parole, modélisation acoustique et linguistique, algorithmes d'entraînement des modèles). J'ai été ensuite en mesure de créer mon propre système de reconnaissance vocale en utilisant l'outil open-source Kaldi. J'ai, par la suite, amélioré les performances du système par l'utilisation de réseaux de neurones et en le rendant robuste au bruit. Enfin, j'ai développé une application de reconnaissance vocale fonctionnant en temps réel avec un microphone.

Abstract

The goal of my internship is to implement Artificial Intelligence for speech recognition. I first had to understand fundamental ideas about Automatic Speech Recognition such as speech processing, statistical acoustic and linguistic modelling, training and decoding algorithms. Then, with this knowledge I managed to build my own ASR system using the open-source Kaldi toolkit. The next step was to improve overall performances of the system by using more advanced techniques such as Deep Neural Networks. I finally created my own speech recognition application using a microphone.

Table des matières

Acronymes

AM Acoustic Model.
ASR Automatic Speech Recognition.

DNN Deep Neural Network.

EM Expectation-Maximization.

fMLLR feature space Maximum Likelihood Linear Regression.

GMM Gaussian Mixture Model.

HMM Hidden Markov Model.

IA Intelligence Artificielle.

LDA Linear Discriminant Analysis.
LM Language Model.

MFCC Mel-Frequency Cepstral Coefficients.
MLLT Maximum Likelihood Linear Transform.

SAT Speaker Adaptative Training.
SNR Signal-to-Noise Ratio.

WER Word Error Rate.
WFST Weighted Finite-State Transducers.

Table des figures

Listings

Liste des tableaux

Chapitre 1

Contexte

1.1 Thales

Thales est un groupe d'électronique spécialisé dans cinq grands secteurs :

- Aéronautique
- Espace
- Transport terrestre
- Identité et Sécurité numériques
- Défense et sécurité

Thales compte 80 000 collaborateurs à travers le monde, répartis dans 68 pays. C'est un acteur clé de la sécurité des citoyens, des infrastructures et des États.
Coté en bourse et avec près de 19Md d'euros de chiffre d'affaires, l'entreprise se place aujourd'hui comme un des leaders mondiaux des technologies et du numérique.

Figure 1.1 – Thales dans le monde

J'ai effectué mon stage dans la branche THALES SIX GTS basée à Gennevilliers sur le site CRISTAL (Hauts-de-Seine). THALES SIX GTS est spécialisée dans les produits et systèmes d'information et de communication sécurisés pour les marchés de la défense, de la sécurité, du transport terrestre ainsi que dans la cybersécurité.

SIX GTS est composée de diverses unités dont le centre de compétence Hardware & Software Technologies HTE.

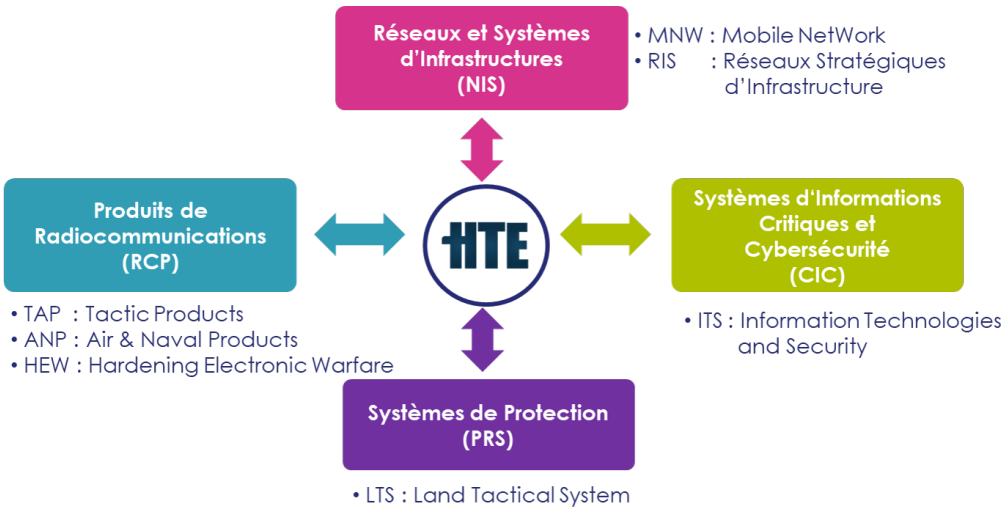

FIGURE 1.2 – Domaines d'activités de HTE

Dans le cadre de mon stage, j'ai intégré l'équipe MMP (*Multi-Media Processing*) du service HTE. Les compétences clés de MMP – dans le contexte des transmissions radio – sont le traitement de la parole et la compression image et vidéo (codage bas-débit).

Le sujet de mon stage s'inscrit dans le service PHONIE du projet CONTACT (*Communications numériques tactiques et de théâtre*) initié par la DGA (*Direction générale de l'armement*). Ce programme stratégique est destiné à équiper la majorité des plateformes des forces armées en postes de radio tactiques de nouvelle génération utilisant une technologie innovante dite de "radio logicielle". [3]

L'amélioration de la qualité audio est au coeur de ce programme. Un système de communication est composé de l'encodage de la parole, de sa transmission puis de son décodage.
La technologie *Speech-to-Text* permet de réduire considérablement la quantité de données à envoyer (en passant d'un signal acoustique à du texte). Elle est donc très intéressante dans un contexte de transmission à très bas débit. A partir du texte, on peut reconstruire la parole à l'aide d'un synthétiseur, c'est le *Text-to-Speech*. Le *Speech-to-Text* est également utilisé pour des applications de commande vocale.

1.2 Objectifs du projet et défis à relever

L'objectif du stage est de réaliser un démonstrateur de système de reconnaissance vocale en temps réel et dans le cadre d'une application à vocabulaire restreint.

Pour mener à bien ce projet j'ai dû :

— Comprendre les concepts fondateurs d'un système de reconnaissance (traitement du signal de la parole, approches statistique et deep-learning, algorithmes d'apprentissage et de décodage ...).

— Prendre en main les outils logiciels utlisés pour créer le système.

— Constituer une base de données pour entraîner et tester le système.

— Construire différents modèles acoustique et linguistique.

— Optimiser les paramètres des modèles.

— Rendre le système robuste au bruit.

— Mettre en place le décodage temps-réel, c'est-à-dire être capable de retranscrire la parole directement depuis un microphone avec un délai quasi-inexistant.

— Créer le démonstrateur en tant que tel.

1.3 Déroulement du projet

Le stage s'est déroulé de Juillet à mi-Décembre 2020. La liste des tâches à effectuer étant bien définie dès le départ, j'ai pu organiser mon travail de manière efficace, logique et cohérente.

Juillet Découverte, lecture de la bibliographie et appropriation du sujet.

Août Premiers essais sur un exemple simple : la reconnaissance de chiffres.

Septembre Contruction de modèles plus complexes, avec de plus grosses bases de données.

Octobre Amélioration du système et ajout de la robustesse au bruit.

Novembre Mise en place du décodage "online" et création de l'interface graphique du démonstrateur.

Décembre Améliorations mineures et finalisation.

J'ai travaillé de manière autonome tout au long du projet, tout en faisant des points régulièrement (1 à 2 fois par mois) avec mes encadrants pour vérifier l'avancée de mes travaux et discuter ensemble des possibles améliorations et des choix pour la suite du projet.

Bien que mon sujet de stage soit indépendant, il reste lié à l'activité de l'équipe PHONIE. J'ai donc eu l'occasion d'aider l'équipe en mettant à contribution mon expertise. Par exemple, avec la réalisation d'une expérience visant à illustrer l'intérêt des stratégies d'augmentation de données à travers une démonstration de reconnaissance vocale (*Proof of Concept*) et en rédigeant la partie du livrable associée.

Chapitre 2

Concepts fondateurs de la reconnaissance vocale

2.1 Introduction

La parole est un phénomène complexe, autant du fait de sa production que de sa perception. Il est commun de penser que la parole est composée de mots eux mêmes composés de phonèmes bien identifiables. En réalité, la parole est un processus dynamique dont les parties constitutrices ne sont pas figées. C'est pour cette raison que les méthodes de description de parole utilisent des modèles probabilistes. Les applications de transcription (*Speech-to-Text*) ne pourront donc jamais atteindre 0% d'erreur. L'objectif reste quand même de s'en approcher au maximum, tout en gardant à l'esprit que l'oreille humaine possède un taux d'erreur d'environ 5%.

Un système de reconnaissance vocale permet de décoder et convertir le signal de parole en un ensemble d'unités linguistique, typiquement une séquence de mots. Au cours des 70 dernières années, la recherche dans le domaine de la reconnaissance vocale a fait des progrès significatifs. On peut dénombrer 5 générations associées aux technologies de leurs époques respectives [6, p. 521] :

1. **Génération 1** (1930 - 1950) : utilisation d'une méthode ad hoc pour reconnaître des sons et des mots isolés issus d'un vocabulaire restreint (2-100 mots).
2. **Génération 2** (1950 - 1960) : utilisation d'une approche acoustique-phonétique pour reconnaître des phonèmes.
3. **Génération 3** (1960 - 1980) : utilisation d'une approche de reconnaissance de *pattern* pour la reconnaissance vocale d'un vocabulaire de taille moyenne (100-1000 mots) ; utilisation de la programmation dynamique afin d'obtenir un alignement temporel des *pattern*.
4. **Génération 4** (1980 - 2000) : utilisation de *Hidden Markov Model (HMM)* pour modéliser la dynamique de la parole ; utilisation des méthodes d'entraînement *forward-backward* et *segmentation K-means* ; utilisation des algorithmes de *Viterbi* ; introduction des *Neural Network (NN)* pour estimer les densités de probabilités a posteriori.
5. **Génération 5** (2000 - 2020) : utilisation de techniques de calculs parallèles pour améliorer la capacité de décision ; combinaison des approches HMM et acoustique-phonétique pour corriger les irrégularités linguistiques ; amélioration de la robustesse au bruit ; combinaison optimale de modèles par *machine learning*.

2.2 Architecture d'un système de reconnaissance vocale

Le système possède quatre composants principaux (figure 2.1) :

— Traitement du signal et extraction des *features* : le signal audio est transformé du domaine temporel au domaine fréquentiel, le vecteur descripteur est ensuite extrait dans un format compatible avec le modèle acoustique.

— Modèle acoustique : contient la connaissance acoustique, c'est-à-dire l'aptitude à reconnaître des phonèmes. À partir d'une séquence de *features* de taille temporelle variable, donne un score aux phonèmes connus du modèle.

— Modèle de langage : donne la probabilité d'une séquence de mots en introduisant la notion de grammaire au système.

— Recherche de la meilleure transcription - Décodage : combinaison des deux modèles. Une suite de phonème reconnue par le modèle acoustique est associée à une suite de mots par l'intermédiaire d'un dictionnaire de prononciation. Cette suite de mots –appelée *hypothèse*– est ensuite confrontée au modèle de langage. On obtient alors le résultat le plus probable d'un point de vue acoustique et linguistique.

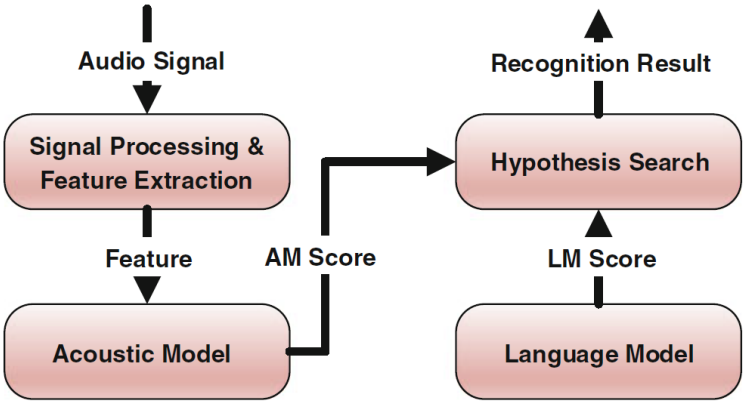

FIGURE 2.1 – Architecture d'un système de reconnaissance vocale [8]

L'objectif du système est de déterminer la phrase la plus probable –dans l'ensemble des phrases du langage– à partir d'une observation acoustique. C'est-à-dire trouver la phrase "responsable" de la génération du signal audio de la parole.

Notons Y cette entrée acoustique avec y_i les tranches temporelles consécutives associées (fenêtres de 25ms décalées toutes les 10ms) telles que :

$$Y = y_1, y_2, y_3, ..., y_t \tag{2.1}$$

De la même manière, notons W une séquence de mots w_i :

$$W = w_1, w_2, w_3, ...w_n \tag{2.2}$$

L'objectif énoncé plus haut se traduit de manière probabiliste comme suit :

$$\hat{W} = \underset{W \in L}{\operatorname{argmax}} P(W|Y) \tag{2.3}$$

$P(W|Y)$ étant difficile à modéliser, on utilise le théorème de Bayes et comme $P(Y)$ ne dépend pas de W, on a :

$$\hat{W} = \underset{W \in L}{\operatorname{argmax}} \frac{P(Y|W)P(W)}{P(Y)} = \underset{W \in L}{\operatorname{argmax}} P(Y|W)P(W) \tag{2.4}$$

Pour résumé, la phrase $\hat{W}$ la plus probable étant donné une observation Y est celle qui maximise le produit des deux probabilités : $P(W)$ est appelée **prior probability**, elle est calculée par le **modèle de langage**. $P(Y|W)$ est appelée **observation likelihood**, elle est calculée par le **modèle acoustique**. [10]

Pour n'importe quel W, le modèle acoustique correspondant est obtenu par le concaténation des modèles de phonèmes, en suivant le dictionnaire de prononciation. Les paramètres de ces modèles sont estimés à partir des signaux audio et de leur transcription. Le modèle de langage est un modèle N-gram, la probabilité d'un mot est conditionné par les N-1 mots précédants. Les paramètres du N-gram sont estimés en comptant les N-tuples présent dans le corpus d'entraînement. La recherche du décodeur s'effectue parmi toutes les séquences de mots possibles, tout en élagant (pruning) les hypothèses les moins probables afin de réduire le temps de calcul. On obtient la séquence de mots la plus probable en sortie lorsque la fin de l'énoncé est atteinte. [6, p. 540]

La suite du chapitre est consacrée à la description plus en détails de ces procédés.

2.3 Traitement du signal de la parole

Cette étape permet d'obtenir une représentation compacte du signal de parole. De manière classique, le vecteur descripteur est calculé toutes les 10 ms sur une tranche de 25ms. Cet *overlapping* confère une redondance de l'information contenue dans le signal et permet donc d'améliorer les performances du système. L'objectif étant d'identifier les parties du signal contenant l'information linguistique. Ainsi, deux locuteurs prononçant la même phrase généreront deux signaux acoustiques différents, cependant les vecteurs descripteurs seront quant à eux similaires puisqu'ils contiennent la même information.

Les descripteurs les plus utilisés sont les *mel-frequency ceptral coefficients* (*MFCC*). Les sons produit par un humain sont filtrés par la forme de son canal vocal (il s'étend de la glotte jusqu'aux lèvres et aux narines). Cette particularité physique se retrouve dans l'enveloppe du spectre de puissance à temps court. Le rôle des *MFCC* est de représenter cette enveloppe. [2]

L'extraction de ces coefficients s'effectue de la manière suivante :

1. Découper le signal en petites trames.
2. Pour chaque trame, calculer le spectre de puissance.
3. Appliquer le *mel filterbank* au spectre de puissance et calculer l'énergie associé à chaque filtre.
4. Prendre le log des énergies.
5. Appliquer la transformée en cosinus discrète (DCT).
6. Garder le nombre de coefficients DCT voulu, ce sont les MFCC.

Il a été prouvé expérimentalement que les 13 premiers coefficients contiennent l'information la plus importante pour la reconnaissance vocale (c_k). Afin de pouvoir déterminer les changements temporels du spectre, les coefficients delta sont inclus au vecteur descripteurs. Ainsi, les y_k introduit dans 2.1 s'expriment :

$$y_k = \begin{pmatrix} c_k \\ \Delta c_k \\ \Delta\Delta c_k \end{pmatrix} \tag{2.5}$$

$$\Delta c_k = c_{k+2} - c_{k-2} \tag{2.6}$$

$$\Delta\Delta c_k = \Delta c_{k+1} - \Delta c_{k-1} \tag{2.7}$$

2.4 Modélisation acoustique

2.4.1 Choix de l'unité linguistique à modéliser

Le modèle acoustisque doit prendre en compte les variations de locuteurs, de contextes et d'environnement. Pour cela, il est essentiel de bien choisir les unités linguistiques que l'on souhaite modéliser. Pour des applications à vocabulaire restreint, on peut considérer de modéliser des mots entiers, il y a cependant deux inconvénients majeurs à cette approche :

1. Il faut disposer de beaucoup de données, en particulier l'enregistrement des mots du vocabulaire avec des variations (on ne dispose pas toujours de telles bases de données).
2. Cette approche n'est plus viable lorsque le nombre de mots à reconnaître devient conséquent.

Une seconde approche consiste à modéliser les parties constitutrices des mots à reconnaître, on modélise alors les *phonèmes*. Cette approche nécessite moins de données et s'adapte beaucoup mieux à des vocabulaires plus large. En fait, à partir de modèles de phonèmes bien entraîné, on est capable de reconnaître des mots qui n'étaient pas présent dans les données d'apprentissage (à condition de connaître la définition phonétique de ces mots).

Cependant, un modèle par phonème (on parle de *modèle monophone* ou *context-independent models*) n'est pas suffisant. En effet, un même phonème peut avoir plusieurs prononciation en fonction de sa position dans le mot. Considérons par exemples les mots suivants :

ENSTA : 3 n s t a

Thales : t a l e s

Le phonème "s" se prononce différemment de part l'influence des phonèmes voisins, c'est-à-dire avant et après le "s". Pour prendre en compte cette variabilité, on utilise le *modèle triphone* (*context-dependant models*). On aura alors autant de modèle par phonème que de combinaisons de voisins possibles. Si l'application comporte N phonèmes, il y aura alors N^3 triphones. En pratique, on ne modèlise que les triphones pertinents (de l'ordre de $10N$ triphones).

2.4.2 Modélisation des unités linguistiques

Le modèle acoustique permet de créer une représentation efficace des séquences de vecteurs descripteurs pour chaque unité linguistique. L'objectif étant d'estimer la probabilité $P(Y|W)$ de la séquence de vecteurs acoustiques Y associé au mot ou phonème W. Chaque nouvelle entrée acoustique est associée à une probabilité de mots/phonèmes.
En pratique, on utilise des modèles statistiques pour calculer la probabilité $P(Y|W)$. De part leur performances et leur implémentation facile, les *Modèles de Markov Cachés* ou *Hidden Markov Models* (HMMs) réprésentent un standard pour la modélisation acoustique.

Hidden Markov Model

Un HMM est une machine à état fini, où la transition d'un état i vers un état j est effectuée à chaque pas temporel avec une probabilité a_{ij}. Contrairement à une chaîne de Markov classique, nous n'avons pas une connaissance direct de l'état courant. Les états sont alors dits "cachés" et ont une probabilité d'émettre une observation. Ainsi, en entrant dans l'état j, un vecteur d'observation y_t est généré en suivant la densité de probabilité $b_j(y_t)$.
Dans un système de reconnaissance basé sur les HMM, on considère que chaque séquence de vecteurs descripteurs acoustiques associée à une unité linguistique est généré par une chaîne de Markov. [8]

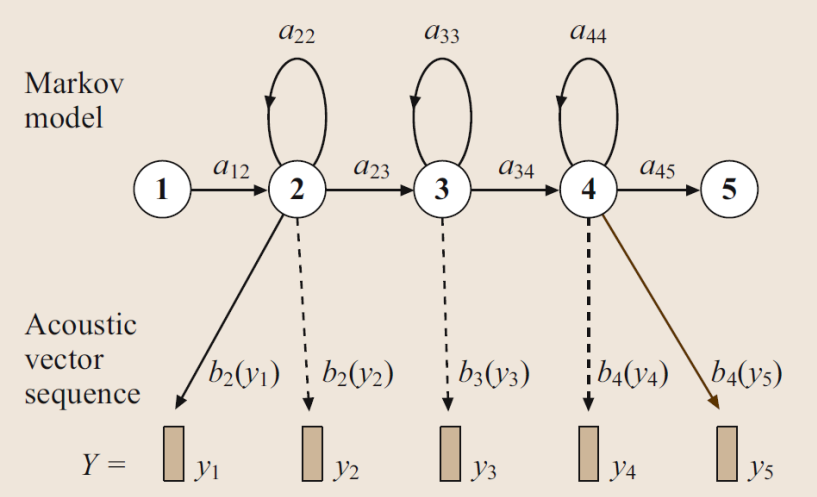

FIGURE 2.2 – Modèlisation phonétique par HMM [6]

Le HMM de la figure 2.2 modèlise une unité linguistique. La topologie du HMM est définie par 5 états de gauche à droite. Les états 1 et 5, situés respectivement au début et à la fin de la chaîne, représentent l'entrée et la sortie. Ils n'ont pas de probabilité d'émission et permettent d'isoler le phonème pour en faciliter la concaténation. On modèlisera alors un mot par la concaténation des modèles HMM de chaque phonème du mot. Les états 2, 3, 4 ont une probabilité d'émission (b_i) pouvant être estimer par un *Gaussian Mixture Model* ou un *Deep Neural Network*. Par chaque état émetteur, le HMM peut rester sur l'état courant ou bien passer au suivant. Cette particularité du HMM lui permet de prendre en compte des dictions plus ou moins lentes du phonème concerné.

Un HMM est défini par un ensemble de paramètres $\Phi = (A, B, \pi)$, avec :

— $A = (a_{ij})$ la matrice stochastique de transition.

— $B = (b_j(y))$ la matrice stochastique des probabilités d'émission avec y le vecteur descripteur.

— $\pi = (\pi_i)$ la matrice d'initialisation des états.

Il y a autant de HMM qu'il y a d'unités linguistiques. On a donc un ensemble $M = \{M_1, M_2, \ldots, M_U\}$ d'unités possibles associées à un ensemble de paramètres $\Phi = \{\Phi_1, \Phi_2, \ldots, \Phi_U\}$. L'objectif du modèle acoustique est d'estimer la probabilité que le modèle M génère la séquence de vecteur acoustique Y.

Pour atteindre cet objectif, trois problèmes majeurs sont à résoudre : [14]

— L'évaluation/estimation : déterminer $P(Y|\Phi)$. On utilise le *Forward Algorithm*.

— Le décodage : trouver la séquence d'état $S = (S_1, S_2, \ldots, S_T)$ la plus probable qui génère la séquence observée $Y = (Y_1, Y_2, \ldots, Y_T)$. On utilise l'algorithme de *Viterbi*.

— L'entraînement : maximiser le produit $\prod_Y P(Y|\Phi)$ en ajustant les paramètres du modèle. On utilise l'algorithme de *Baum-Welch*.

2.4.3 Modèle conventionnel : HMM - Gaussian Mixture Model

Le premier candidat pour estimer les fonctions de densité de probabilité $b_j(y)$ est le *multivariate Gaussian mixture*. Son avantage majeur réside dans sa capacité à approximer n'importe quelle distribution. Avec M *Gaussian mixture*, on a : [17]

$$b_j(y) = \sum_{k=1}^{M} c_{jk} N(y, \mu_{jk}, \Sigma_{jk}) = \sum_{k=1}^{M} c_{jk} b_{jk}(y) \tag{2.8}$$

où $b_{jk}(y)$ désigne une Gaussienne de vecteur moyen μ_{jk} et de matrice de covariance Σ_{jk} pour l'état j, les c_{jk} sont des poids.

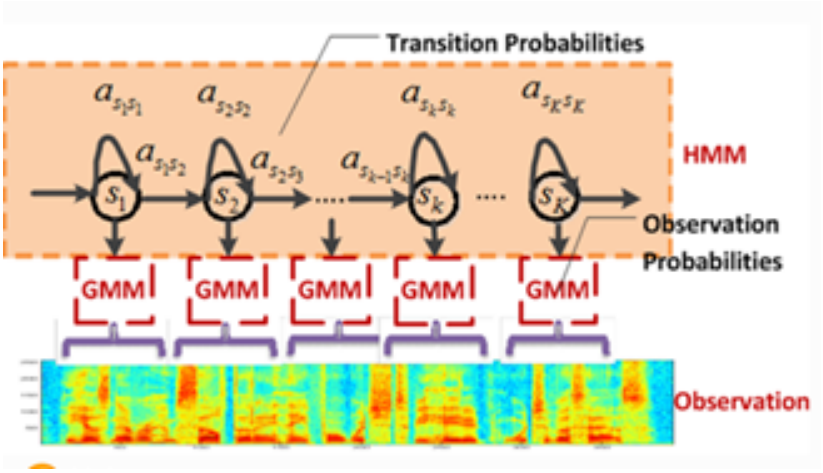

FIGURE 2.3 – Architecture d'un système HMM-GMM [8]

Les modèles HMM-GMM ont longtemps été considérés comme étant les meilleurs. Cela est dû à leur capacité d'adaptation à des signaux acoustiques variables et à l'efficacité de l'algorithme d'*Expectation-Maximization* pour l'estimation de ses paramètres. Cependant, ces modèles considèrent une distribution de données gaussienne. Il s'agit là d'une approximation grossière. De ce fait, pour modéliser des données hautement non-linéaires, les GMM requièrent un nombre gigantesque de paramètres.

2.4.4 Modèle hybride : HMM - Deep Neural Network

Une solution pour contourner les limites du GMM est le **réseau de neurones**. En effet, le DNN possède une capacité de classification bien supérieure à celle du GMM. Ainsi, le DNN modélise des frontières non-linéaires entre les différentes classes de sorties, qui sont ici les états cachés du HMM.

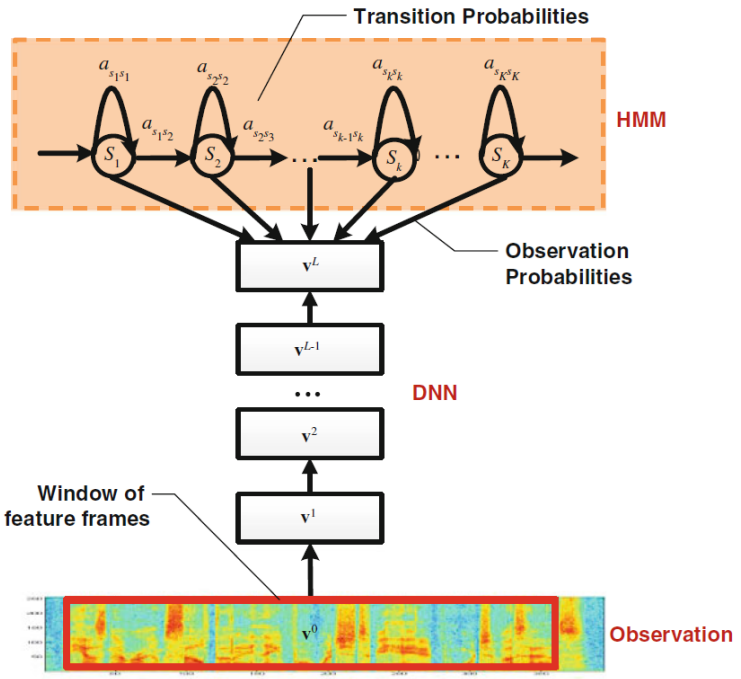

FIGURE 2.4 – Architecture d'un système HMM-DNN [8]

On utilise une approche supervisée pour entraîner le réseau de neurones. Les données d'entraînement labélisées correspondent aux alignements du modèle HMM-GMM, c'est-à-dire la relation entre les états cachés (que l'on veut déterminer) et les frames émises (que l'on observe).
Les différents choix d'architecture, d'algorithme d'optimisation, de fonctions d'activation et d'hyperparamètres feront l'objet d'une discussion approfondie dans les chapitres suivants (voir 5.1.2).

2.5 Modélisation linguistique

Le modèle acoustique permet la reconnaissance de mots à partir du signal acoustique. Il s'agit maintenant d'apporter au système la syntaxe de la langue, les règles qui déterminent si une phrase a du sens et est juste grammaticalement.

Le modèle de langage est exprimé par la distribution $P(W)$ à partir de W ; rendant compte de la fréquence à laquelle la chaîne de mots W apparait dans une phrase. On peut décomposer $P(W)$: [17]

$$\begin{aligned} P(W) &= P(w_1, w_2, \ldots, w_n) \\ &= P(w_1)P(w_2|w_1)P(w_3|w_1, w_2)\ldots P(w_n|w_1, w_2, \ldots, w_{n-1}) \\ &= \prod_{i=1}^{n} P(w_i|w_1, w_2, \ldots, w_{i-1}) \end{aligned} \tag{2.9}$$

avec $P(w_i|w_1, w_2, \ldots, w_{i-1})$ la probabilité que w_i apparaisse en tenant compte la séquence $w_1, w_2, \ldots, w_{i-1}$

En réalité, une telle probabilité est impossible à estimer car il existe des séquences de mots uniques. Une solution pratique est de restreindre la séquence aux n mots précédants, on parle de modèle de langage **n-gram**.
Ainsi, si le mot dépend des deux mots précédants nous avons un *tri-gram* : $P(w_i|w_{i-2}, w_{i-1})$. Le *tri-gram* est particulièrement efficace puisqu'il est suffisamment long pour rendre compte de la grammaire mais également suffisamment court pour être estimé facilement.

Pour le calculer, on a besoin de compter le nombre de pairs $C(w_{i-2}w_{i-1})$ et de triplets $C(w_{i-2}w_{i-1}w_i)$ du corpus d'apprentissage :

$$P(w_i|w_{i-2}, w_{i-1}) = \frac{C(w_{i-2}w_{i-1}w_i)}{C(w_{i-2}w_{i-1})} \tag{2.10}$$

2.6 Décodeur

Le but du décodeur est de déterminer la suite de mots ayant la probabilité la plus élevée étant donné le lexique et les modèles acoustique et linguistique [9]. Comme indiqué dans l'équation 2.3, la phrase la plus probable $\hat{W}$ – pour le vecteur descripteur $Y = y_1, ..., y_t$ –, est trouvée en cherchant toutes les séquences d'états associées à toutes les séquences de mots et en concordance avec la séquence d'observation acoustique. L'espace de recherche obtenu pour une application à vocabulaire large est énorme et nécessite des algorithmes efficaces. En pratique, on utilise des *Weighted Finite-State Transducers (WFST)*. Cette approche est décrite dans la section 3.2.5.

Chapitre 3

Réalisation d'un système de reconnaissance vocale

3.1 Présentation de l'outil Kaldi

3.1.1 Définition et motivation

Kaldi est un outil open-source de reconnaissance vocale écrit en C++. L'objectif de Kaldi est de fournir du code flexible et "facile" à comprendre. Il est alors possible de modifier le code source pour répondre à des besoins spécifiques et rajouter des fonctionnalitées. Les points forts de Kaldi sont la taille de sa communauté ainsi que la grande variété d'algorithmes et de recettes [1] disponible. Le *toolkit* est constamment mis à jour pour suivre *l'état de l'art* du domaine de la reconnaissance vocale. De plus, son *framework* bien pensé et robuste permet aux utilisateurs de gagner en productivité en se concentrant uniquement sur leur système de reconnaissance. Le *framework* garantit également que le système restera fonctionnel dans le temps. Il existe naturellement d'autres alternatives open-source telles que *Julius, Sphinx ou HTK*, toutes inférieures à Kaldi en terme de performance et de support. C'est pour les raisons citées ci-dessus, que Kaldi a été retenu pour le développement du système de reconnaissance vocale.

3.1.2 Historique du projet et limites

Le projet Kaldi a été initié en 2009 par l'université Johns Hopkins, dans leur laboratoire *"Low Development Cost, High Quality Speech Recognition for New Languages and Domains"*. Bien que performant, l'âge du *toolkit* commence à se faire ressentir. L'ensemble des techniques développées au cours de la dernière décennie étant disponible. Il est aisé de se perdre dans la multitude de possibilités qu'offre Kaldi et d'utiliser une recette (ou des étapes de recette) obsolète par inadvertance.
Un autre inconvénient est la difficulté de la prise en main de Kaldi. En effet, c'est un outil destiné aux experts de la reconnaissance vocale, il faut donc avoir une bonne maîtrise des divers éléments présentés dans le chapitre 2 avant de pouvoir passer à l'action.

Ce constat est à l'origine du projet *SpeechBrain* [4] qui vise à créer une boite à outils à la fois simple d'utilisation tout en conservant les performances atteintes par Kaldi. *SpeechBrain* a l'ambition de créer un outil polyvalent unique basé sur le package Python *PyTorch* pour réaliser toutes sortes de tâches.

1. Une recette est une succession d'étapes élémentaires permettant la création d'un système de reconnaissance.

3.1.3 Fonctionnement général

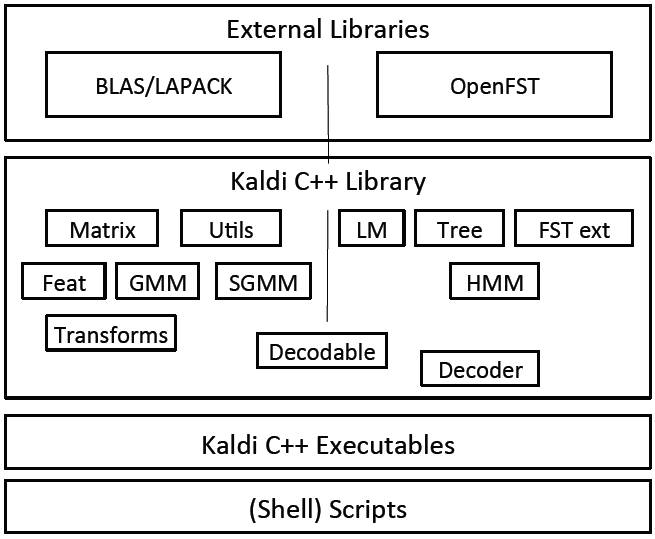

FIGURE 3.1 – Vue simplifiée des différents composants de Kaldi. [12]

External Libraries Le finite-state framework de OpenFst. "Basic Linear Algebra Subroutines" (BLAS) et "Linear Algebra PACKage" (LAPACK) pour le calcul algébrique.

Kaldi Library Les bibliothèques externes sont utilisées dans le code C++ de Kaldi pour implémenter les diverses briques technologiques.

Kaldi Executables Ces briques sont ensuite assemblées pour donner lieu à des fonctionnalités.

Scripts Shell Les fonctionnalités permettent enfin de composer le système de reconnaissance vocale.

3.1.4 Structure de Kaldi

```
kaldi-trunk/
    cmake/
    docker/
    egs/
    misc/
    scripts/
    src/
    tools/
    windows/
README.md
```

egs Contient les exemples. Un exemple est un système de reconnaissance adapté à *une base de données* et élaboré à partir *d'une recette*.

src Contient le code source de Kaldi (Kaldi Libraries et Kaldi Executables).

tools Contient les outils externes utilisé par Kaldi (Openfst, LAPACK).

3.1.5 Structure d'un exemple d'application

```
example/
    conf/
    data/
        dev/
        test/
        train/
        lang/
    exp/
    local/
    mfcc/
    steps/
    utils/
    run.sh
```

conf Contient les fichers de configuration pour le décodage et l'extraction des descripteurs.

data Contient les sets de données au format Kaldi, voir 3.2.1. *lang/* contient le lexique de l'application, il s'agit de la décomposition phonémique des mots à reconnaître.

exp Les résultats de chaque recette sont enregistrés sous forme *d'expérience*.

mfcc Contient les mfcc des données fournies dans *data/*.

steps et utils Contiennent une grande variétée de script shell utilisés pour la création de recettes.

run.sh Contient une recette.

3.2 Elaboration d'une recette Kaldi classique

L'objectif de cette section est de présenter et d'expliquer de manière générale les différentes étapes nécessaires à la création d'un système de reconnaissance sur Kaldi. À partir de données d'entraînement, nous pouvons entraîner un modèle statistique de la parole. Le modèle donne la probabilité qu'une phrase donnée produise un signal acoustique. À partir d'un signal acoustique, nous pouvons déterminer la phrase la plus probable.

3.2.1 Préparation des données

Pour fonctionner, nous devons fournir à Kaldi les ressources suivantes :

— Un corpus labélisé : Il contient un ensemble d'enregistrement de parole avec les transcriptions associées.

— Un dictionnaire de prononciation : Aussi appelé *lexicon*. Chaque mot possède une décomposition en phonème.

— Du texte pour entraîner le modèle de langage : La concaténation des transcriptions du corpus peut suffire.

Il existe de nombreuses bases de données, à choisir en fonction de l'application désirée. La plupart sont disponibles à partir du *Linguistic Data Consortium* mais sont payantes (de quelques centaines à quelques milliers de dollards).

Parmi les plus populaires on peut citer :

— **Fisher** : Plusieurs milliers d'heures de conversation téléphonique (transcritent à la main !)

— **Globalphone** : Ensemble de phrases lues en français, cette base de données ne contient qu'une trentaine d'heures de parole.

— **Timit** : Ensemble de phrases lues en anglais, cependant la transcription est uniquement phonémique. Il s'agit de la base de données référence pour la reconnaissance de phonème.

Pour obtenir un **bon modèle**, il faut (condition nécessaire mais pas suffisante) de **bonnes données**. C'est-à-dire des données en quantité suffisante, et représentative du contexte d'utilisation désiré. On cherchera à avoir le plus de variabilité possible. Cela peut être au niveau de la diversité des locuteurs (ethnie, âge, sexe), ou bien du contexte (parole lue, parlée, variation du débit).
Une fois la base de données choisit, elle est découpée en 2 à 3 sets disjoints :

— Train set : Le set d'entraînement nécessite le plus de données, en général 60 à 80 % de la totalité des données disponible. Plus la base de données est grande, plus on peut réduire la taille du *train test* au profit des autres sets.

— Test set Le set de test contient des données inconnues du modèle pour en évaluer la capacité de généralisation.

— Validation set : Ce set est utilisé pour l'entraînement de modèles *Deep Neural Network*. Par ailleurs, l'utilisateur n'est pas obligé de créer ce set lui-même. Kaldi extrait le *validation set* à partir du *train set*. Le set de validation permet de juger de la qualité de l'apprentissage en cours de celui-ci en émettant un diagnostique sur la manière dont on entraîne le modèle. Cela permet de maximiser la capacité de généralisation en limitant le surapprentissage du modèle.

3.2.2 Extraction des descripteurs

Une fois la base de données constituée, il convient d'extraire l'information utile du signal de la parole. Comme expliqué dans la section 2.3, le vecteur descripteur permet l'obtention d'une représentation compacte du signal acoustique. On utilise généralement les *MFCC*. Le vecteur est ensuite normalisé.

Grâce au fichier de configuration, on peut définir le nombre de coefficients cepstraux du vecteur descripteur (en général entre 13 et 20). On peut également choisir de garder le terme énergétique.

3.2.3 Création du modèle de langage

Le modèle de langage est généré par *SRILM - The SRI Language Modeling Toolkit*. Cet outil permet la création de modèles de langage statistiques. Il suffit de lui fournir du texte représentatif de l'application cible (par exemple les transcriptions du train set) ainsi que l'ordre du n-gram désiré. Il se charge de calculer les probabilités log d'apparition de séquences de k mots avec $k \in [\![1, n]\!]$. Le résultat est donné sous la forme d'un fichier .arpa, qui sera transformé par la suite au format .fst par l'intermédiaire d'un script Kaldi.

3.2.4 Création du modèle acoustique

Nous traitons ici uniquement du modèle *HMM-GMM*, puisqu'il s'agit de la technologie du modèle de référence. Les alternatives à cette modélisation sont présentées dans le chapitre 5. Le modèle est entraîné à partir des données audio du train set.

Entraînement et alignement des monophones

Le modèle *monophone* est un modèle acoustique qui ne contient pas d'information contextuelle sur les phonèmes – le précédent et le suivant –. Il s'agit d'une brique élémentaire qui servira ensuite à créer le modèle *triphone*.

Le processus d'entraînement du modèle (estimation des paramètres des HMM associés à chaque phonème) est un processus itératif composé de deux phases :

1. Entraînement des gaussiennes associées à chaque phonème grâce à l'algorithme *Expectation Maximization*. On obtient ensuite la transcription phonétique du signal audio utilisé.
2. Alignement de la transcription avec le signal audio. Cette étape permet d'associer un phonème à un son.

À chaque itération, la phase d'alignement permet de cibler les portions audio correspondant aux divers phonèmes à reconnaître. La phase d'entraînement gagne alors en efficacité puisque l'on définit précisement les données utiles à l'entraînement de chaque phonème.

Pour cette étape d'entraînement des monophones, on fixe le nombre de gaussiennes pour la modélisation ainsi que le nombre d'itérations.

Entraînement et alignement des triphones

Le modèle *triphone* est introduit pour prendre en compte la variabilité de la prononciation des phonèmes dans leur contexte et ainsi améliorer grandement les performances de reconnaissance. Comme expliqué dans la section 2.4.1 pour N phonèmes on a N^3 triphones possibles. En pratique, il n'est pas possible –et surtout inutile, puisque seulement une partie des triphones sont présents dans le corpus d'apprentissage– de modéliser tous les triphones. Il faut, par ailleurs, que ces triphones apparaissent plusieurs fois dans les données d'apprentissage pour calculer des statisques. Un arbre de décision phonétique se charge ensuite de regrouper les triphones en un nombre limité d'unités linguistiques distinctes (les racines de l'arbre sont des *context dependant phones* et les feuilles sont des *context independant phones*). On réduit ainsi grandement le nombre de paramètres du modèle.

La stratégie itérative d'entraînement du modèle triphone est la même que pour le modèle monophone, à ceci près qu'elle utilise maintenant les alignements issus du modèle monophone.

Ici, le concepteur renseigne le nombre de feuilles de l'arbre de décision, le nombre de gaussiennes et le nombre d'itérations d'entraînement.

3.2.5 Décodeur

Weighted Finite-State Transducers

Pour réprésenter les divers composants du système de reconnaissance (HMM, modèles de dépendence du contexte, dictionnaire de prononciation et grammaire), Kaldi utilise des *weighted finite-state transducers (wfst)*. Un *finite-state transducer* est un automate dont les transitions sont marquées par des symboles d'entrée et de sortie. Ainsi, un chemin dans le *tranducer* associe une séquence de symboles d'entrée à une séquence de symboles de sortie. Le *weighted tranducers* ajoute des poids aux transitions en plus de ces symboles. Ces poids peuvent être interprétés comme des probabilités. [11]

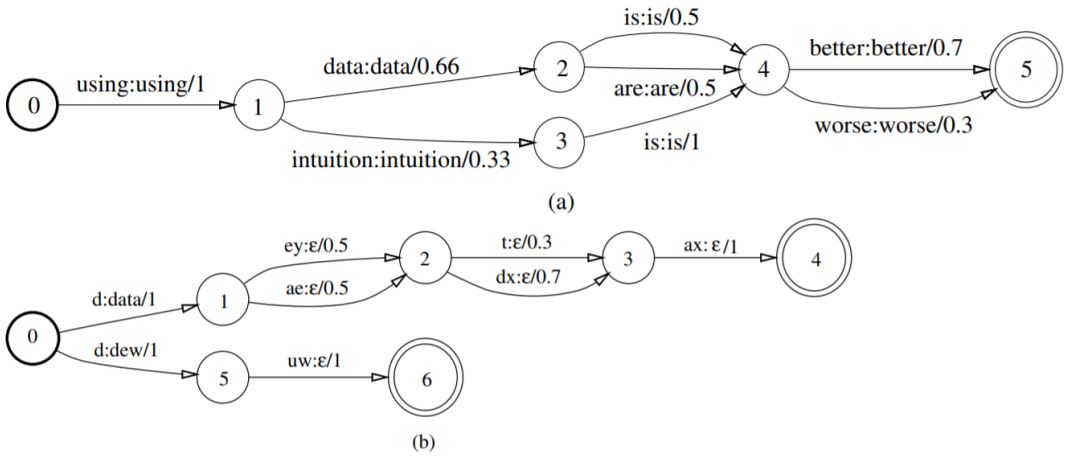

FIGURE 3.2 – Exemples de wfst. (a) wfst de la grammaire ; (b) wfst de la prononciation [11]

La figure 3.2.a est un exemple de weighted FST représentant le modèle de langage. Les symboles d'entrée et de sortie sont des mots, les poids sont les probabilités du modèle de langage. La probabilité d'une phrase sera alors le produit des poids correspondants. La figure 3.2.b est un exemple de weighted FST représentant le dictionnaire de prononciation des mots *data* et *dew*.

Construction du graphe de décodage

On désire construire un graphe global qui contient toute l'information des composants du système, chaque composant étant lui-même représenté par un FST :

- **G** : le modèle de langage (grammaire), les symboles d'entrée et de sortie sont des mots.
- **L** : le dictionnaire de pronocniation (lexique), les symboles de sortie sont des mots et les symboles d'entrée sont des phonèmes.
- **C** : l'arbre de décision phonétique (contexte), les symboles de sortie sont des phonèmes et les symboles d'entrée sont des *context dependant phones*.
- **H** : la topologie des HMMs des phonèmes, les symboles de sortie sont des *context dependant phones* et les symboles d'entrée sont des *transitions id* qui encodent entre autres le *pdf id* (identifiant des densités de probabilités associées aux états cachés du HMM). [1]

On obtient le graphe final **HCLG** par composition.

Décodage

Le graphe HCLG ainsi construit est trop gros pour être directement décodé via l'algorithme standard de *Viterbi*. On utilise alors une *beam search* qui consiste à élaguer (*prune*) les états du graphe ayant un score inférieur au meilleur score moins le *beam*. Il y a donc un compromis entre vitesse de décodage (*beam* grand) et précision (*beam* petit).

3.3 Méthodologie et critères d'évaluation

On cherche maintenant à évaluer les performances du système de reconnaissance. Cette évaluation est très importante puisqu'elle permet une comparaison relative des modèles. Le critère le plus utilisé est le *Word Error Rate* ou *WER*. On calcule le WER des divers modèles à partir des phrases du test set.

Il y a trois types d'erreurs possibles entre la phrase à reconnaître et la phrase reconnue :

— *Substitution :* Le mot à reconnaître est remplacé par un autre mot.

— *Deletion :* Le mot à reconnaître n'est pas repéré, il a été "oublié".

— *Insertion :* Le système a reconnu un mot supplémentaire, qui n'est pas contenu dans la phrase à reconnaître.

Le WER d'une phrase est défini par :

$$WordErrorRate = \frac{S + D + I}{N} \tag{3.1}$$

avec S le nombre de substitution, D le nombre de suppression, I le nombre d'insertion et N le nombre de mots contenus dans la phrase correcte.

La source d'erreur la plus commune est la substitution. Cela est dû à la similarité phonétique d'un grand nombre de mots de la langue française. Considérons par exemple 2 mots de 5 phonèmes ayant 4 phonèmes en commun. La distinction entre ces 2 mots sera déterminée par le phonème restant, le système aura alors du mal à choisir entre ces 2 mots. La seule manière de contourner ce problème est d'avoir une modélisation très fine des phonèmes.

Chapitre 4

Mise en oeuvre et évaluation d'un système de référence

Le système de référence est un système de reconnaissance à vocabulaire large, faisant office de base solide pour la suite du projet. L'objectif de ce chapitre est d'expliquer, étape par étape, comment ce système a été conçu.

4.1 Présentation et préparation de la base de données

4.1.1 Globalphone

GlobalPhone est une base de données polyglotte developpée par XLingual en collaboration avec le *Karlsruhe Institute of Technology*. *GlobalPhone* contient des signaux de parole lus, de très bonne qualité ainsi que leurs transcriptions et est disponible en une grande variété de langues dont le français. Cette base de données est idéale pour le déploiement rapide d'un système de reconnaissance à vocabulaire large. La version française de *GlobalPhone* est composée de 100 locuteurs, prononçant chacun 100 phrases d'environ 20 mots, sur une durée totale de 30 heures. Les locuteurs sont d'âge et de sexe variés ; les phrases lues sont issues de journal "Le Monde" et traitent d'une grande variété de sujets.

Le caractère lu des phrases implique que le système de reconnaissance sera plus performant pour des signaux de paroles dont la diction est plutôt lente et monotone. La commande vocale - qui est notre objectif final- répond à ces critères et justifie donc l'utilisation de la base de données *GlobalPhone* pour l'entraînement des modèles acoustiques.

4.1.2 Préparation en amont

GlobalPhone est fourni avec trois dossiers :

— wav : contient les fichiers audio

— spk : contient des informations sur les locuteurs (âge, sexe, fumeur, rhume, allergie ...)

— trl : contient les transcriptions

Malheureusement le format des données n'est pas directement compatible avec celui de Kaldi. Il y a donc une étape de préparation préliminaire réalisée par un script Python. De plus, les sets de test, entraînement et validation ne sont pas encore créés.

Format d'origine

Ci-dessous un extrait du fichier FR001.trl contenant les transcriptions concernant le premier locuteur.

```
;SprecherID FR001
; 1:
A ces trois sortes de jours sont associés deux prix par jour en fonction de la
consommation en heures pleines ou creuses
...
```

Le fichier .wav associé est de la forme :

```
FR001_1.adc.wav
```

Format compatible Kaldi

Le script Python génère le dossier data que l'on fournit à Kaldi :

```
data/
    train/
        text
        utt2spk
        wav.scp
    test/
        text
        utt2spk
        wav.scp
    dev/
        text
        utt2spk
        wav.scp
```

Ce qui donne par exemple pour la première phrase du premier locuteur :

```
text : FR001_1 a ces trois sortes de jours sont associes deux prix par jour
en fonction de la consommation en heures pleines ou creuses

wav.scp : FR001_1 /home/marc/Documents/BDD/Global_phone/wav/FR001_1.adc.wav

utt2spk : FR001_1 FR001
```

Les sets de validation et de test sont composés de 8 locuteurs chacun. Tous les autres locuteurs sont dans le set de train.

Dictionnaire de prononciation

Il faut par ailleurs fournir à Kaldi le dictionnaire de prononciation des mots de GlobalPhone. Le dictionnaire contient environ 20 000 mots. Par exemple :

```
abaissement         a b 3 s m a -
abandon         a b a - d o -
abandonne         a b a - d C n
```

Corpus d'apprentissage de la langue

Il faut enfin générer le fichier texte utilisé pour le modèle de langage. Il s'agit ici de la concaténation de toutes les phrases lues de Globalphone.

4.2 Entraînement des modèles avec Kaldi

Le script Bash complet est disponible en annexe. On met ici l'accent sur les commandes clés.

Paramètres globaux

```
nj=15
nj_decode=8
lm_order=3
exp_dir=exp
```

On renseigne le nombre de *jobs* à lancer en parallèle pour réduire le temps de calcul. On utilise un tri-gram pour le modèle de langage. Les résultats sont placés dans le dossier /exp.

4.2.1 Extraction des descripteurs et normalisation

```
steps/make_mfcc.sh --mfcc-config conf/mfcc.conf --nj $nj --cmd "$train_cmd" data/
    train $exp_dir/make_mfcc/train $mfccdir
steps/compute_cmvn_stats.sh data/train $exp_dir/make_mfcc/train $mfccdir
```

On utilise le script Kaldi *make_ mfcc.sh* pour calculer les mfccs des sets de train, test et dev. Les résultats sont stockés dans des fichiers archives ".ark".

Le script *compute_ cmvn_ stats.sh* effectue la "Cepstral mean and variance normalization". Cette étape de normalisation permet d'augmenter la robustesse de la reconnaissance en limitant l'influence de l'environnement de la prise de son et des caractéristiques du locuteur.

4.2.2 Création du modèle de langage

```
utils/prepare_lang.sh data/local/dict "<UNK>" data/local/lang data/lang
ngram-count -order $lm_order -write-vocab $local/tmp/vocab-full.txt -wbdiscount -
    text data/text.txt -lm $local/tmp/lm.arpa
arpa2fst --disambig-symbol=#0 --read-symbol-table=$lang/words.txt $local/tmp/lm.
    arpa > $lang/G.fst
```

Le script *prepare_ lang.sh* génère le dossier /lang à partir du dictionnaire de prononciation. /lang permet ensuite la création de L.fst, qui contient le lexique.

On utilise ensuite la commande "ngram-count" de SRILM pour génèrer le fichier .arpa à partir du corpus text.txt. Le fichier arpa tri-gram donne les probabilités log associées à des triplet de mots :

```
-0.5228788        traitement automatique des
-0.1760913        circulation automobile et
-0.1760913        des autorisations de
```

En réalité, le fichier arpa contient le 1-gram, 2-gram et 3-gram pour un total de 172 785 combinaisons. C'est le nombre de combinaisons distinctes de 1, 2 et 3 mots contenues dans text.txt.

Enfin, cette information est mise sous forme de fst grâce à la commande "arpa2fst". On génère ainsi G.fst, qui contient la grammaire.

4.2.3 Entraînement, décodage et alignement des monophones

```
#training
steps/train_mono.sh --nj $nj --cmd "$train_cmd" data/train data/lang $exp_dir/
    mono || exit 1

#decoding
utils/mkgraph.sh --mono data/lang $exp_dir/mono $exp_dir/mono/graph || exit 1
steps/decode.sh --config conf/decode.config --nj $nj_decode --cmd "$decode_cmd" \
 $exp_dir/mono/graph data/test $exp_dir/mono/decode_test

#alignment
steps/align_si.sh --nj $nj --cmd "$train_cmd" data/train data/lang $exp_dir/mono
    $exp_dir/mono_ali || exit 1
```

Pour l'entraînement des monophones, on utilise les paramètres par défaut. C'est-à-dire un total de 1000 gaussiennes et 40 itérations d'entraînement. Le nombre d'itération est largement suffisant pour assurer la convergence des algorithmes et offrir un premier modèle acoustique. Le dossier /lang contient toute l'information relative aux phonèmes, dont la topologie des HMMs.

Il y a 234 phonèmes dans GlobalPhone. Un phonème est associé à un HMM, lui-même composé de 3 ou 5 états cachés en fonction de la nature du phonème (silence ou non-silence phone). La probabilité d'émission d'un état est modélisée par un GMM. Cependant, certains états sont communs entre les HMMs et ne nécessitent donc pas un modèle propre. On parle alors d'états liés ou tied-states. Le nombre total de GMM à modéliser sera alors inférieur au nombre total d'états cachés.

Pour obtenir une répresentation compacte de cette information, Kaldi introduit la notion de *Triples*. Un triple est défini par {phones, état, pdf id}, Pdf désigne une *probability density function* (la probabilité d'émission). Ainsi, les 234 phonèmes sont représentés par $224*3+10*5=722$ triples. Grâce aux *tied-states* on réduit le nombre de GMM à entrainer de 722 à 178.

Pour la phase de décodage, on crée dans un premier temps le graphe relatif aux HMM (H.fst) puis on fait la composition avec les graphes du modèle de langage. Le FST de contexte C.fst est simplifié pour le modèle monophone et ne contient que l'information sur la position du phonème dans le mot (début, milieu, fin) sans tenir compte des voisins. Le graphe composé HCLG.fst est enfin placé dans le dossier /mono/graph.

Pour tester le modèle, on décode les données du set de test. Les résultats seront discutés dans la section suivante.

Enfin, le script *align_si.sh* génère les alignements monophone utilisés pour l'entraînement du second modèle acoustique, le modèle triphone.

4.2.4 Entraînement, décodage et alignement des triphones

```
numLeavesTri1=3100
numGaussTri1=50000

#training
steps/train_deltas.sh --cmd "$train_cmd" $numLeavesTri1 $numGaussTri1 data/train
   data/lang $exp_dir/mono_ali $exp_dir/tri1 || exit 1

#decoding
utils/mkgraph.sh data/lang $exp_dir/tri1 $exp_dir/tri1/graph || exit 1
steps/decode.sh --config conf/decode.config --nj $nj_decode --cmd "$decode_cmd" \
 $exp_dir/tri1/graph data/test $exp_dir/tri1/decode_test

#alignment
steps/align_si.sh --nj $nj --cmd "$train_cmd" --use-graphs true \
 data/train data/lang $exp_dir/tri1 $exp_dir/tri1_ali
```

L'entraînement du modèle triphone suit la même logique que le monophone. On entraîne d'abord les gaussiennes grâce au script *train_ deltas.sh* en renseignant cette fois le nombre de feuilles de l'arbre de décision ($numLeavesTri1) qui correspond au nombre de "context independant phones", ainsi que le nombre de gaussiennes contenues dans les GMMs ($numGaussTri1).

Il est à noter que le paramètre le plus important est $numLeavesTri1, puisqu'il définit indirectement le nombre de triphones que l'on va modéliser. Si il est trop petit, on ne s'éloigne pas vraiment du modèle monophone. Au contraire, si il est trop grand, nous n'aurons pas assez de données pour obtenir un modèle performant.

Nous avons ici 9453 triples contre 722 pour le modèle monophone, puisque désormais la probabilité d'émission d'un état est modélisé par plusieurs GMM pour prendre en compte les variations de prononciation dépendant des phonèmes voisins. Ainsi, un phonème très présent dans la langue, sera modélisé par de nombreux triphones correspondants aux différents contextes dans lequel il est prononcé. Il y aura donc un nombre conséquent de GMMs pour modéliser les émissions des états cachés associées au HMM de ce phonème. Le modèle triphone comporte 2424 pdfs contre 178 pour le modèle monophone. L'arbre de décision détermine quel pdf choisir parmi ceux disponibles pour un état.

Les étapes de décodage et d'alignement sont similaires à celles du modèle précedant. L'alignement triphone sera utilisé par la suite, pour labéliser les données d'entraînement du réseau de neurones.

4.3 Evaluation

Les résultats présentés ici correspondent aux meilleurs obtenus après l'essai de multiples configurations de paramètres des modèles acoustique et linguistique.

À l'issue de l'étape de décodage, on obtient plusieurs *WER* correspondant à diverses pondérations des modèles acoustique et linguistique. Considérons à titre d'exemple deux phrases prononcées par deux locuteurs différents.

```
;SprecherID FR091
; 1:
Il faut adresser un certificat au juge des tutelles du lieu de résidence
du patient s'il prend lui même l'initiative de la demande

;SprecherID FR098
; 1:
L'abaissement des charges sociales pèse sur les emplois les moins qualifiés
dans les pays où le coût indirect du travail est élevé
```

4.3.1 Résultats du modèle monophone

```
compute-wer --text --mode=present ark:exp/mono/decode_test/scoring/test_filt.txt ark,p:-
%WER 20.17 [ 4478 / 22196, 159 ins, 1413 del, 2906 sub ]
%SER 95.13 [ 781 / 821 ]


FR091_1 il faut adresser un certificat au juge de tutelle de la duree dans des
patients supprimant initiative de la demande

FR098_1 l abaissement des charges sociales pese sur les entourent les moins
qualifies dans les pays ou le cout indirect du travail etat civil
```

4.3.2 Résultats du modèle triphone

```
compute-wer --text --mode=present ark:exp/tri1/decode_test/scoring/test_filt.txt ark,p:-
%WER 9.11 [ 2021 / 22196, 127 ins, 422 del, 1472 sub ]
%SER 81.61 [ 670 / 821 ]


FR091_1 il faut adresser un certificat au juge des tutelles du milieu de
residence de patience le prend le meme initiative de la demande

FR098_1 l abaissement des charges sociales pese sur les emplois le moins
qualifies dans les pays ou le cout indirect du travail est elevee
```

Chapitre 5

Améliorations du système

Le système de référence étant maintenant clairement défini et fonctionnel, nous nous intéressons désormais aux possibles améliorations. En particulier, la définition d'un modèle acoustique plus performant grâce à des tranformations des descripteurs et un réseau de neurones ; la robustesse du système au bruit ; la capacité d'extraire un sous-système à vocubulaire restreint inclus dans le vocabulaire large de référence.

5.1 Améliorations du modèle acoustique

Le modèle triphone du système de référence a donné des résultats convaincants, il est cependant possible d'améliorer les performances en utilisant des techniques de transformation des descripteurs couplées à un réseau de neurones.

Les améliorations du système sont incrémentales, au sens où chaque nouveau modèle utilise les alignements du précédent. On entraîne le modèle triphone de référence à partir du modèle monophone, puis on améliore la référence avec des techniques de transformation et enfin on entraîne le modèle DNN à partir de ce modèle amélioré.

5.1.1 Amélioration des descripteurs

Transformation linéaire

Au lieu d'utiliser directement les vecteurs descripteurs tels que les MFCCs, on peut les transformer dans un nouvel espace afin de restreindre les effets de l'environnement, de distortion et du locuteur. On améliore ainsi les performances de la reconnaissance.

La solution retenue consiste en une transformation LDA (*Linear Discriminant Analysis*) et une estimation MLLT (*Maximum Likelihood Linear Transform*).

— LDA : Il s'agit d'une transformation linéaire qui vise à décorréler et à réduire la dimension du vecteur descripteur en entrée.

— MLLT : Estime les paramètres d'une transformée linéaire afin de rendre les descripteurs modélisés par GMM plus précis.

Adaptation au locuteur

Une autre source d'erreur réside dans les différences de locutions. On utilise alors des techniques d'adaptation au locuteur pour modifier les paramètres du modèle acoustique afin de mieux correspondre à un locuteur cible et ainsi améliorer les performances de reconnaissance.

Cette adaptation est réalisée par fMLLR (*feature space Maximum Likelihood Linear Regression*). Il s'agit d'un ensemble de matrices de transformation utilisé pour modifier les gaussiennes des HMM. L'avantage de cette technique est qu'elle ne nécessite pas beaucoup de données. On peut donc d'abord entraîner un modèle général indépendant du locuteur (*Speaker Independant*) puis l'adapter à partir de quelques phrases par locuteur (*Speaker Dependant*).

Il est également possible d'intégrer l'adaptation au locuteur dans la phase d'entraînement, on parle alors de SAT (*Speaker Adaptative Training*).

Résultats

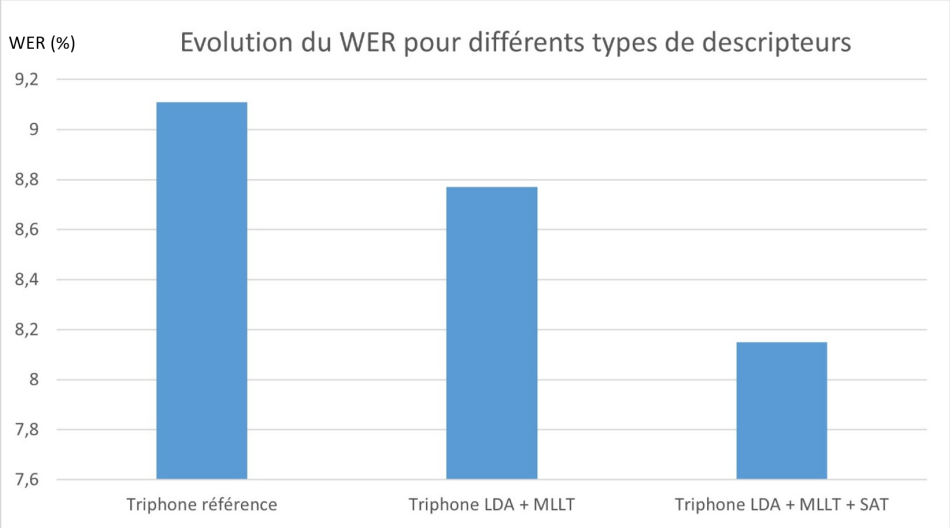

FIGURE 5.1 – Evolution du WER pour différents types de descripteurs

Le WER obtenu avec les descripteurs transformés et l'adaptation au locuteur est de 8,15%, soit une diminution de l'erreur relative de 11,8% par rapport au modèle de référence et de 7,6% par rapport au modèle non adapté au locuteur (figure 5.1).

5.1.2 Modèle acoustique hybride HMM-DNN

Motivation

Dans ce modèle acoustique (2.4.4), le réseau de neurones est utilisé en remplacement des GMM pour modéliser les probabilités d'émission des états cachés du HMM. Autrement dit, le modèle acoustique estime la distribution $p(x|y)$, qui est la probabilité d'observer le descripteur acoustique x conditionnée par le *context dependant HMM state* y. [5]

En fait, le réseau ne modélise pas directement $p(x|y)$ mais $p(y|x)$. Ce qui permet de voir le DNN comme *classifier* des états cachés du HMM (sortie) avec en entrée le vecteur descripteur acoustique. On utilise ensuite la loi de Bayes pour se ramener au cadre HMM : $p(x|y) = \frac{p(y|x)p(x)}{p(y)}$.

Labelisation des données d'entraînement

Les données sont labelisées à partir des alignements du modèle HMM-GMM. On associe à chaque phrase du train set une séquence d'états cachés avec la trame du signal temporel correspondante. On calcule ensuite pour chaque trame le vecteur descripteur et on obtient enfin les couples vecteur descripteur-état caché (x, y), utilisés pour entraîner le réseau.

Extraction des descripteurs

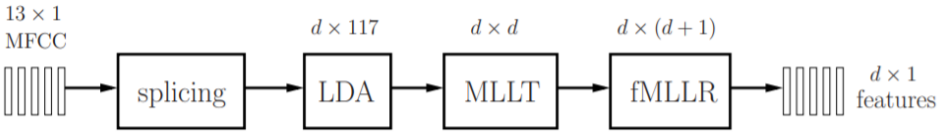

FIGURE 5.2 – Génération du vecteur descripteur [13]

Les MFCC de dimension 13 sont concaténés (*splice*) avec un contexte de 9 frames temporelle (4 à gauche et 4 à droite) pour produire un vecteur de taille 117. Les *features* sont ensuite décorrélées et adaptées en utilisant les techniques décrites plus tôt (figure 5.2). D'après les fondateurs de Kaldi [13], $d = 40$ donne les meilleurs résultats.

Architecture du réseau et entraînement

La fonction d'activation utilisée pour apporter de la non-linéarité est de type *p-norm*. Il s'agit d'une généralisation de la non-linéarité *maxout* ($y = \max_i x_i$). *p-norm* est définie par :

$$y = \|x\|_p = (\sum_i |x_i|^p)^{\frac{1}{p}} \tag{5.1}$$

où le vecteur x désigne un petit groupe d'entrées (en général 5 ou 10). Il a été montré que la valeur $p = 2$ donne les meilleurs résultats. [16].

La non-linéarité réduit la dimension : On part d'un vecteur de taille 3000 (*pnorm input*) que l'on décompose en 300 groupes de 10 (*pnorm output*). On applique *p-norm* à chacun de ces petits groupes et on obtient un vecteur de taille 300.

Le réseau de neurones utilisé est composé de 4 *hidden layers*. Le *output layer* est un *softmax layer* dont la dimension correspond au nombre de *context dependant state*, soit 2480 dans notre cas. Le nombre total de paramètres est de 4538480.

Le réseau a été entraîné sur 10 epochs avec un learning rate variable de 0.01 à 0.001 puis 5 epochs avec un learning rate constant de 0.001. Le learning rate est choisi plutôt bas car on dispose de beaucoup de données et donc suffisament de temps pour assurer la convergence. Les *hidden layers* sont ajoutés au fur et à mesure de l'entraînement (1 par epoch).

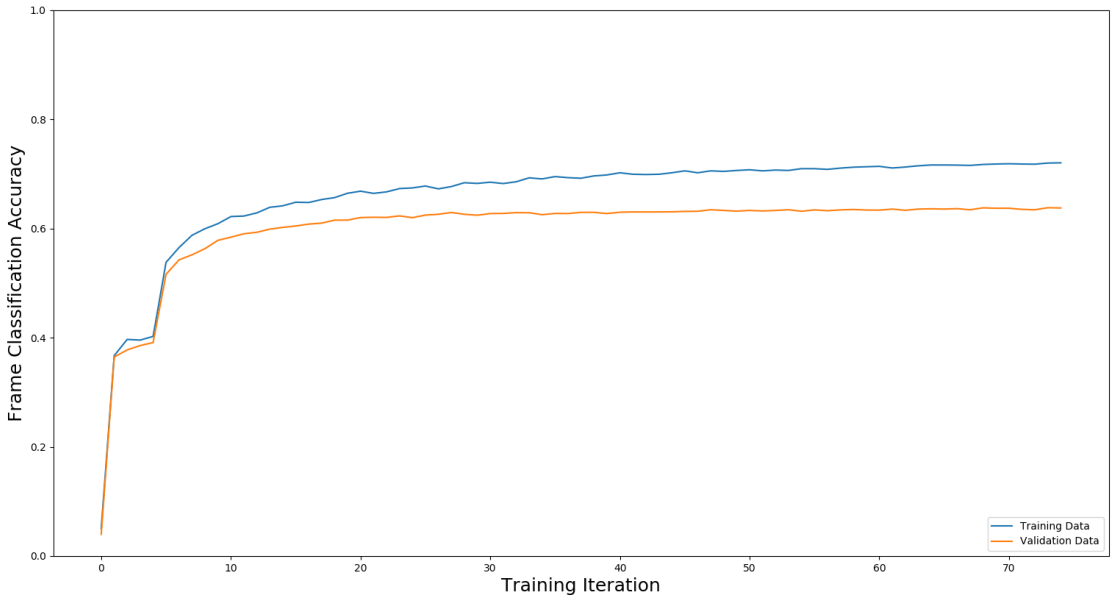

FIGURE 5.3 – Evolution de la précision en fonction du nombre d'itérations

Les bonds en précision au début de l'entraînement correspondent à l'introduction des *hidden layers* (figure 5.3 itérations 5, 10 et 15). Le set de validation permet d'estimer la capacité de généralisation de notre modèle au cours de l'entraînement, en le testant avec des données inconnues. Lorsque la précision sur le *validation set* est constante, on considére que l'entraînement est terminé puisque le système ne s'améliore plus. En continuant l'entraînement, on risque même l'effet inverse avec du sur-apprentissage.

Il faut garder à l'esprit que la précision de la classification renseigne uniquement sur les performances de la modélisation acoustique (une précision de 1 étant le maximum) et que la labélisation des données n'est pas parfaite étant elle-même issue d'un autre modèle. La reconnaissance acoustique est ensuite complétée par la connaissance de la langue apportée par le modèle de langage.

Evaluation

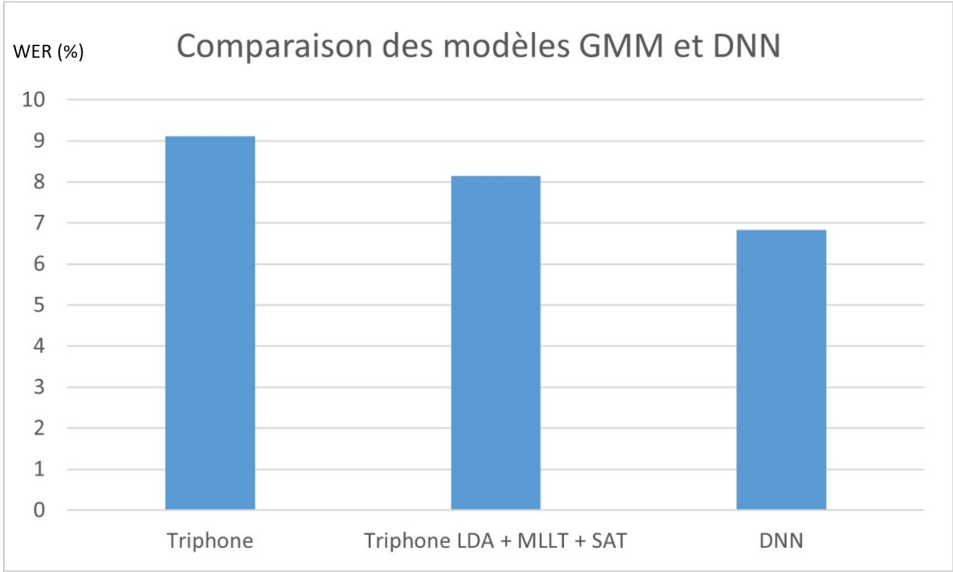

FIGURE 5.4 – Comparaison des modèles GMM et DNN

Le WER obtenu par le modèle DNN est de 6,83% soit une diminution relative de l'erreur de 25% par rapport au modèle triphone de référence et 16,2% par rapport au modèle triphone amélioré. (figure 5.4)

5.2 Robustesse au bruit

Jusqu'à présent les données audio utilisées pour entraîner et tester les modèles n'étaient pas perturbées par l'environnement sonore. Nous nous intéressons donc dans cette section à l'influence du bruit sur les performances du système de reconnaisance. L'objectif étant d'entraîner un nouveau modèle robuste au bruit.

5.2.1 Préparation des données bruitées et création d'un nouveau modèle

Les signaux de bruits retenus correspondent à un contexte militaire avec des bruits de véhicules (avion, bus et train).

Quantification du bruit

Pour générer les données bruitées, on ajoute le signal de bruit aux données d'entrée non-bruitées. Le critère pour quantifier la proportion de bruit dans le signal résultant est le *signal-to-noise ratio* (*SNR*). Le SNR est défini par le rapport de la puissance du signal d'entrée sur la puissance du bruit, la puissance d'un signal étant la moyenne de son énergie :

$$SNR = \frac{P_{signal}}{P_{noise}} \tag{5.2}$$

$$SNR_{dB} = 10 \log_{10}(SNR) \tag{5.3}$$

Afin de restreindre l'influence des très basses fréquences du bruit dans le calcul de la puissance (zone de forte énergie), nous appliquons un filtre passe-haut de fréquence de coupure 150 Hz (voir figure 5.5). En effet, l'énergie des basses fréquences augmente la puissance du signal sans être perceptible par l'oreille humaine. La modification du SNR aurait alors peu d'impact lors de l'écoute du signal de sortie.

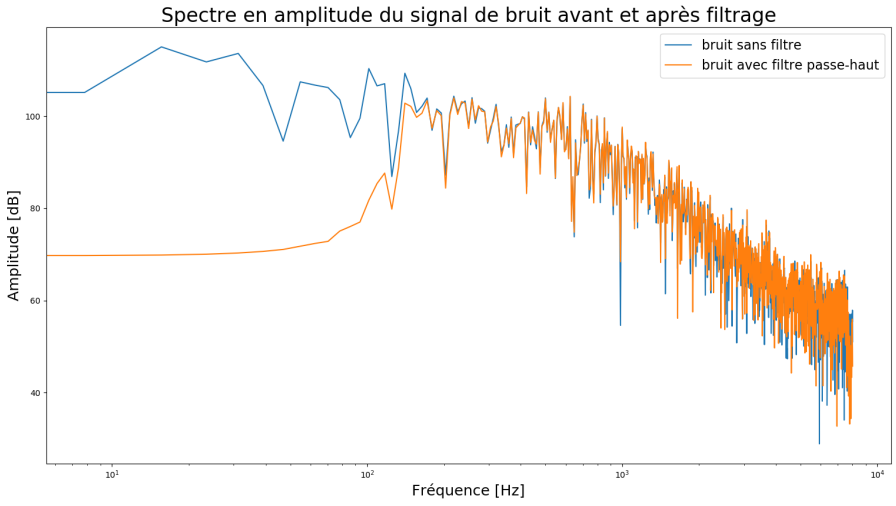

FIGURE 5.5 – Spectre en amplitude (dB) du signal bruit avant et après filtrage

En faisant varier la valeur du SNR cible, nous sommes en mesure de générer des signaux de sortie avec différents niveaux de bruits. Plus il y a de bruit, plus il est difficile de distinguer la parole.

Génération de la base de données bruitée

La base de données bruitée est composée de signaux non-bruités et de signaux bruités dont le SNR varie entre -5 et 20. Pour un SNR de -5, il est très difficile même pour un humain de reconnaître la parole. Pour un SNR de 20, le bruit est difficilement remarquable. Les valeurs des SNR sont en décibels ; lorsque le SNR diminue de 3 dB, le bruit est 2 fois plus fort. Le spectrogramme de la figure 5.6 illustre ce phénomène.

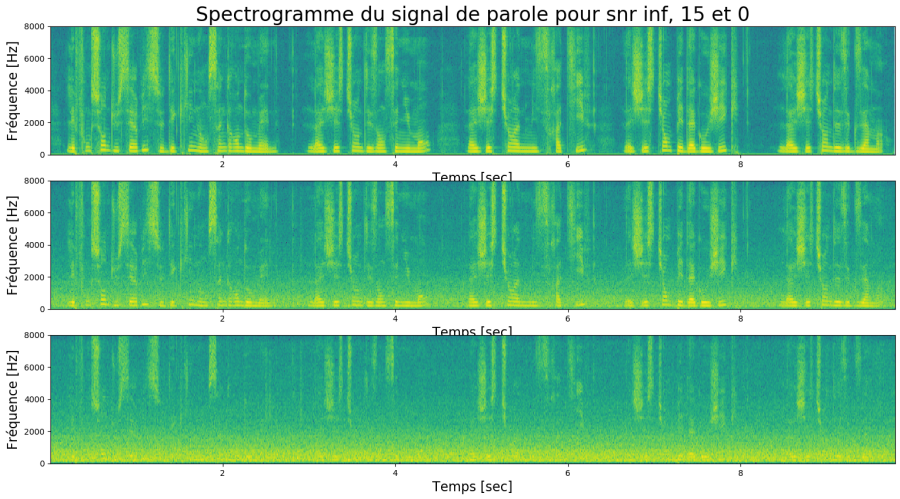

FIGURE 5.6 – Spectrogramme du signal de parole pour snr inf (en haut), 15 (au milieu) et 0 (en bas)

Le programme Python utilisé pour générer les données bruitées prend en entrée les signaux non-bruités et une liste de signaux de bruits (ici, bruit dans un avion, un bus et un train). Pour chaque signal non-bruité on choisit un bruit et un SNR cible entre -5 et 20 en suivant une loi uniforme. Le grand nombre de données assure une répartition harmonieuse des SNR .

$$y_{out} = y_{in} + 10^{Snr - Snr_{cible}} y_{bruit} \tag{5.4}$$

Avec y_{out} le signal bruité résultant, y_{in} le signal d'entrée non-bruité, y_{bruit} le signal de bruit.

Entraînement du modèle bruité et test

Un nouveau modèle acoustique est entraîné à partir de cette nouvelle base de données bruitées. La recette Kaldi utilisée est la même que pour le système de référence.

Pour tester les performances du modèle, on génère plusieurs sets de test correspondant respectivement à des SNR de $-5, 0, 5, 10, 15, 20, inf$.

5.2.2 Comparaison et conclusion

La figure 5.7 vise à comparer les WER des modèles entraînés avec et sans les données bruitées pour les technologies *triphone* et *DNN*.

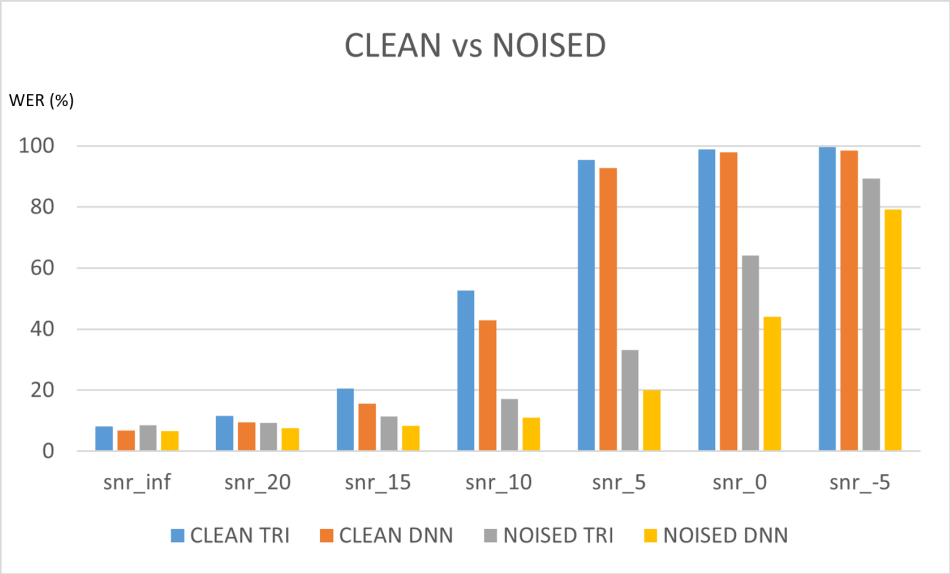

FIGURE 5.7 – Comparaison des modèles bruités et non-bruités pour différents snr

On remarque que les performances des modèles bruité et non-bruité sont semblables pour des niveaux de bruits bas (SNR haut). En revanche, lorsque le bruit commence à être perceptible (SNR 10), les performances du modèle de référence se dégrade considérablement avec environ 1 mot sur 2 reconnus alors que le modèle bruité conserve des performances honorables avec 1 mot sur 10 reconnus.

De manière générale, le modèle bruité possède des performances nettement supérieures au modèle non-bruité.

	CLEAN		NOISED	
	TRI	DNN	TRI	DNN
snr_inf	8,15	6,83	8,62	6,55
snr_20	11,59	9,55	9,35	7,5
snr_15	20,58	15,62	11,33	8,32
snr_10	52,56	42,86	17,06	11,01
snr_5	95,42	92,7	33,23	20,04
snr_0	98,93	97,95	64,01	44,1
snr_-5	99,71	98,49	89,23	79,16

TABLE 5.1 – Récapitulatif des différents WER (SNR en dB, WER en %)

5.3 Application à vocabulaire restreint

L'objectif ici est de pouvoir créer une application à voculaire restreint à partir du modèle de référence. Nous n'avons pas besoin de réentraîner le modèle acoustique à chaque nouvelle application puisqu'il s'agit d'une modélisation phonétique. Nous pouvons simplement réutiliser cette information phonétique. Il n'est donc pas nécessaire d'avoir des enregistrements spécifiques des mots à reconnaître ; le dictionnaire de prononciation couplé aux modèles acoustiques de phonèmes suffisent. Il faut cependant être en mesure de générer le modèle de langage à chaque application puisque le vocabulaire change.

5.3.1 Génération du fichier arpa

Pour générer le modèle de langage du système de référence nous avons utilisé l'outil *SRILM*. Cependant, cette technique de génération du fichier arpa est sur-dimmensionnée par rapport aux besoins d'une application à vocubulaire restreint où l'apparition des mots est équiprobable. J'ai donc écrit un script Python prenant en entrée la liste de mots du vocabulaire et générant en sortie le fichier arpa correspondant.

Par exemple, pour un vocabulaire cible composé des chiffres de zéro à neuf en anglais :

```
\data\
ngram 1=12

\1-grams:
-0.301030       </s>
-99        <s>
-1.301030       zero
-1.301030       one
-1.301030       two
-1.301030       three
-1.301030       four
-1.301030       five
-1.301030       six
-1.301030       seven
-1.301030       eight
-1.301030       nine

\end\
```

Il s'agit d'un modèle 1-gram puisqu'il n'y a pas de lien entre les mots. Entre chaque mot, il y a un silence d'où une probabilité log de $-0.301030 = \log(\frac{1}{2})$. Le reste du temps, il y a un des N mots avec la probabilité $-1.301030 = \log(\frac{1}{2N})$. On génère par la suite G.fst à partir de ce fichier.

5.3.2 Nouveau système sur Kaldi

Notre application est composé de mots, lesquels sont composés de phonèmes. Il faut maintenant être capable d'associer les identifiants des phonèmes de la nouvelle application avec ceux de la référence. Kaldi supporte cette fonctionnalité (généralement utilisé pour rajouter des phonèmes et ainsi agrandir le vocabulaire). Il suffit de donner à Kaldi l'ancien fichier qui contient les phonèmes (*phones_gp.txt* ici) et le nouveau dictionnaire de prononciation.

```
utils/prepare_lang.sh --phone-symbol-table data/phones_gp.txt data/local/dict "<
   UNK>" data/local/lang data/lang
```

En substituant les anciens fichiers phones.txt avec le nouveau que l'on vient de générer. On est capable d'accéder uniquement à l'information acoustique utile du système de référence.

```
for file in $(find . -name phones.txt);
do
  cp -f data/lang/phones.txt $file
done
```

Il ne reste plus qu'à créer les nouveaux graphes HCLG.fst pour les modèles monophone et triphone.

```
utils/mkgraph.sh --mono data/lang $exp_dir/mono $exp_dir/mono/graph || exit 1

utils/mkgraph.sh data/lang $exp_dir/tri1 $exp_dir/tri1/graph || exit 1
```

Pour tester l'application, il faut possèder des enregistrements des mots à reconnaître ou bien tester directement en parlant dans le microphone. On donnera plus de détails sur cette deuxième option dans le chapitre suivant.

Chapitre 6

Optimisation du système pour la réalisation d'un démonstrateur

L'ojectif de ce chapitre est d'adapter le système de reconnaissance afin de décoder la parole en temps réel directement depuis l'entrée audio du microphone, dans une optique de commande vocale.

6.1 Objectifs et contraintes

On souhaite créer une application qui permettrait à l'utilisateur de se déplacer sur une image et de l'annoter par commande vocale. On peut imaginer le contexte d'utilisation suivant : un soldat renseigne l'emplacement de cibles sur une carte tout en garder les mains libres.

La contrainte forte est ici le temps nécessaire pour décoder la parole. Le système doit être réactif, sans pour autant dégrader les performances. Pour répondre à ces exigences, nous développons un nouveau modèle acoustique, plus compact et utilisant des technologies adaptées.

6.2 Décodage temps réel "en ligne"

Le meilleur setup pour décoder en temps réel utilise un réseau de neurone. La stratégie utilisée est de faire l'adaptation au locuteur directement dans le réseau de neurones. Les descripteurs que l'on fournit au réseau sont des MFCC de haute résolution (on garde 40 coefficients contre 13 auparavant) non-adaptés et non-normalisés, auquels on ajoute un *iVector*. Un *iVector* est un vecteur de dimension 100 à 200 qui représente les caractéristiques du locuteur. Le *iVector* apporte au réseau tout ce dont il a besoin de savoir sur le locuteur. Le contexte temporel retenu est plus petit que celui décrit dans la section 5.1.2 avec 7 frames, 3 à gauche et 3 à droite.

6.2.1 Entraînement d'un nouveau réseau de neurones

Le réseau de neurones utilisé est un *pnorm neural network*. Afin de répondre à la contrainte de temps, on réduit le temps de calcul nécessaire à la classification avec une architecture de réseau plus petite que celle décrite dans 5.1.2. Ici, on utilise seulement 2 *hidden layers* avec *pnorm input = 1000* et *pnorm output = 200*. La procédure d'entraînement et les données utilisées ne changent pas (corpus GlobalPhone).

6.2.2 Stratégie de décodage

Le graphe HCLG utilisé pour le décodage reste le même que celui de modèle 5.1.2. La différence réside dans les données d'entrée, on décode le signal de parole par morceaux (*chunks*) de 0,2 à 0,3 sec. Les paramètres de l'algorithme de Viterbi sont aussi adaptés avec un rayon de recherche (*beam*) plus petit pour réduire le temps de calcul et ainsi être en mesure de décoder le *chunk* courant avant l'arrivée du suivant.

Les performances de décodage vont dépendre de la machine utilisée. En effet, dans le même temps, nous sommes en mesure d'explorer une plus grande partie de l'espace de recherche avec un ordinateur puissant, qu'avec un système embarqué. Il faut donc adapter le système de reconnaissance au *hardware* utilisé. Le respect du critére temps-réel se fera à l'insu des performances de reconnaissance.

6.2.3 Comparaison des modèles "offline" et "online"

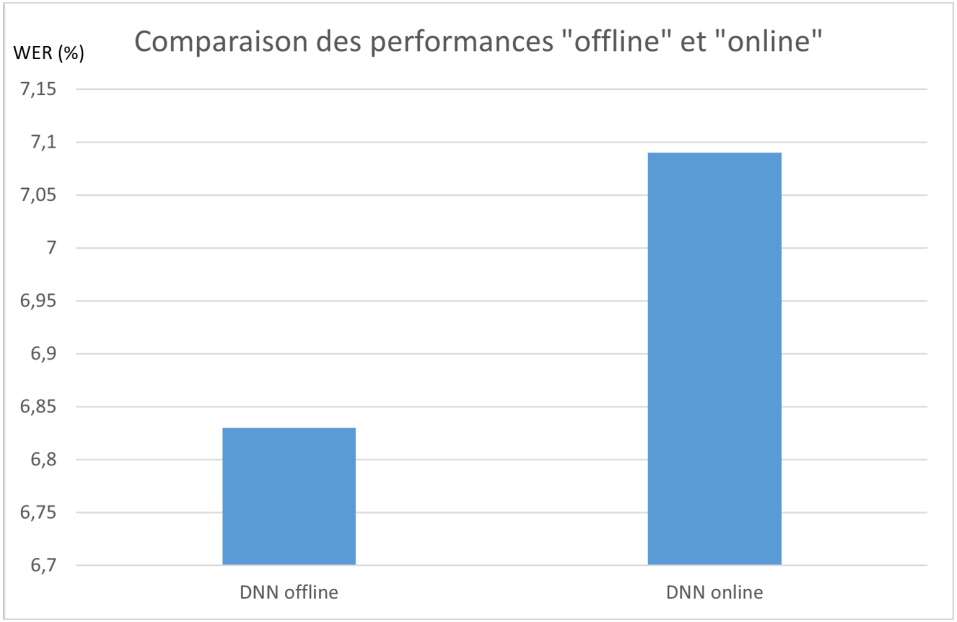

FIGURE 6.1 – Comparaison des performances "offline" et "online"

L'évaluation est faite à partir du set de test de Globalphone. Les performances observées dans la figure 6.1 sont très similaires. Le WER passe de 6,83% pour le mode offline à 7,09% pour le mode online, soit une augmentation relative du taux d'erreur de 3,6%. Ce faible écart s'explique par la qualité du hardware utilisé pour ces tests.

6.3 Présentation du démonstrateur

6.3.1 Description du vocabulaire restreint utilisé

Le vocabulaire restreint utilisé pour le démonstrateur comporte des mots-clés de commande, l'alphabet militaire et des mots de description. Ce sont les seuls mots reconnaissables par le système. On peut cependant modifier le vocabulaire en fonction du contexte d'utilisation (voir 5.3).

```
oui         w i
non         n o -
retour        r t u r
ecrit        e k r i
annotation         a n o t a s j o -
arriere        a r j 3 r
initialisation         i n i s j a l a s j o -
zero    z e r o
un         e -
deux         d 0
trois         t r u a
quatre         k a t r
cinq    s e -
six         s i
sept         s 3 t
huit         q i t
neuf         n 9 f
zoom         z u m
alpha         a l f a
bravo   b r a v o
charlie S a r l i
delta   d e l t a
echo    e k o
foxtrot f C k s t r o t
avion        a v j o -
arbre        a r b r
batiment        b a t i m a -
camion        k a m j o -
terrain         t e r e -
```

6.3.2 Menu

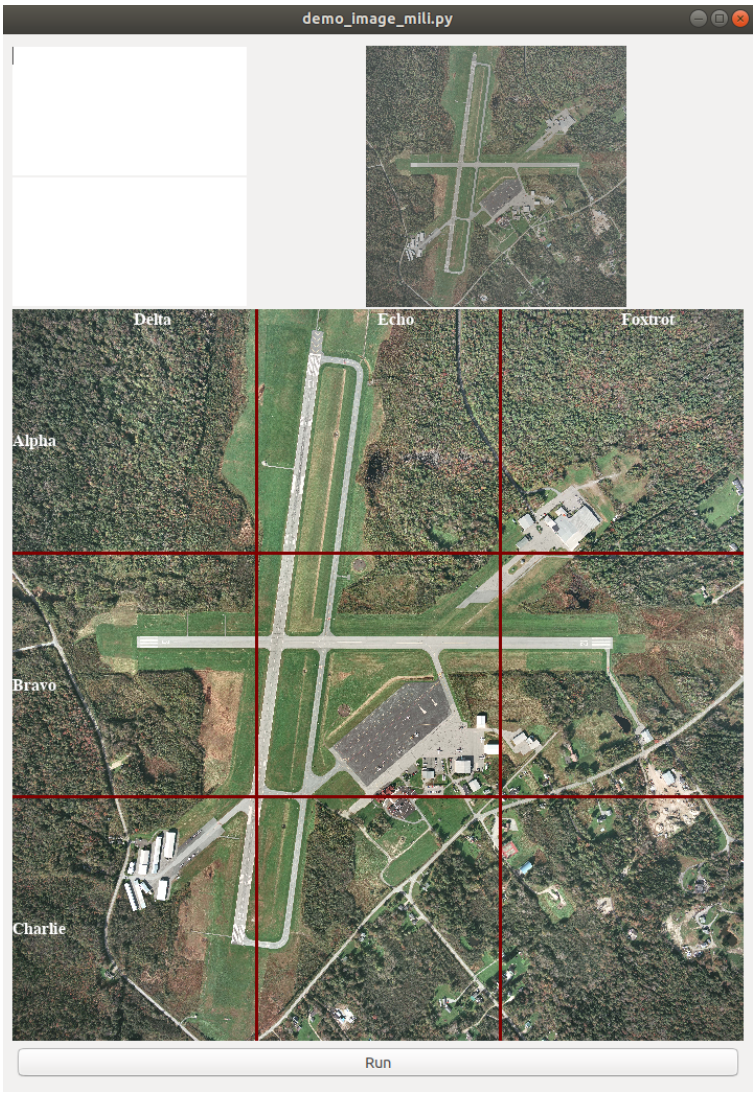

FIGURE 6.2 – Menu de l'application

L'application est composée de :

— Deux *text buffers* en haut à gauche (un pour afficher la commande reconnue, l'autre pour afficher les annotations).

— Une carte pour se localiser sur l'image globale (en haut à droite).

— L'image sur laquelle on peut se déplacer par zoom successifs. L'image est découpée en 9 cases, on repère les cases avec l'alphabet militaire. Alpha-Delta représente par exemple la première case (en haut à gauche).

Pour lancer l'application, on appuit sur **Run**. Afin de limiter l'influence de l'environnement, on utilise un système *push-to-talk* pour décoder la parole uniquement au moment de l'énonciation de la commande.

6.3.3 Zoom

Pour zoomer, on utilise la commande **zoom** suivie des coordonnées de la case sélectionnée. On peut également revenir à l'image précédente avec la commande **zoom arriere**. Ici, on se déplace sur la case centrale par la commande **zoom bravo echo**.

FIGURE 6.3 – Commande de zoom

Une fois la commande reconnue, on coche la case correspondante pour apporter une validation visuelle.

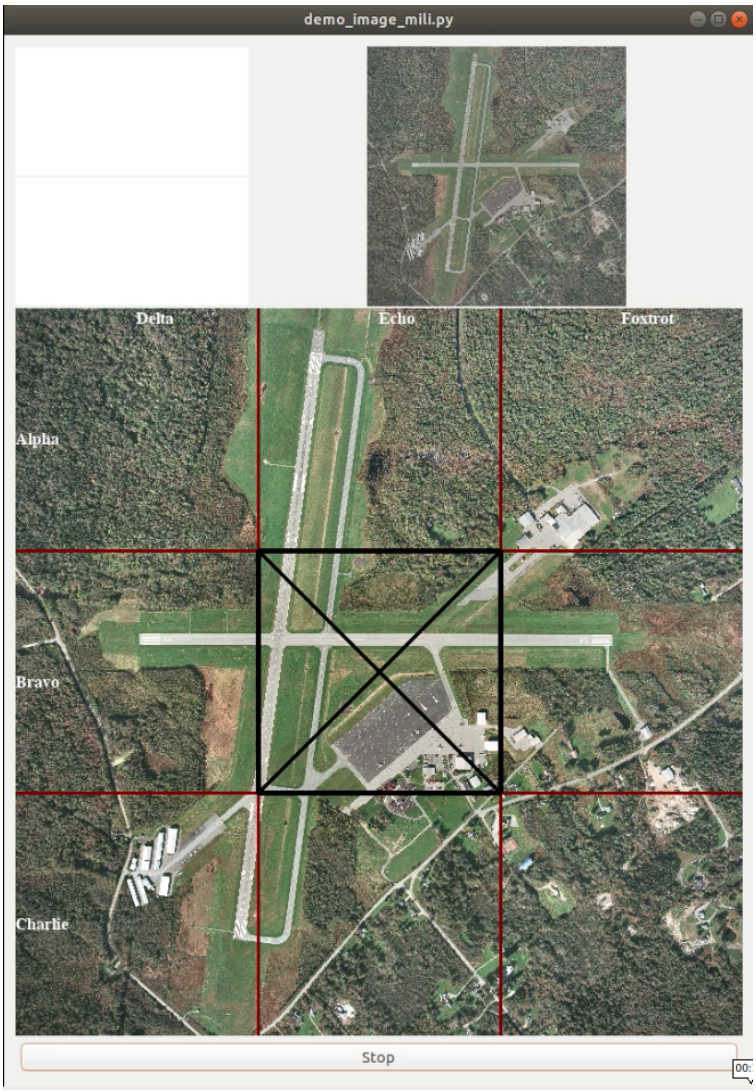

FIGURE 6.4 – Validation visuelle

On affiche la nouvelle image, qui correspond à la case sélectionnée. On peut alors continue à se déplacer si le zoom n'est pas suffisant. Par ailleurs, on peut se repérer sur la carte de localisation, le carré rouge correspondant à l'image couramment affichée.

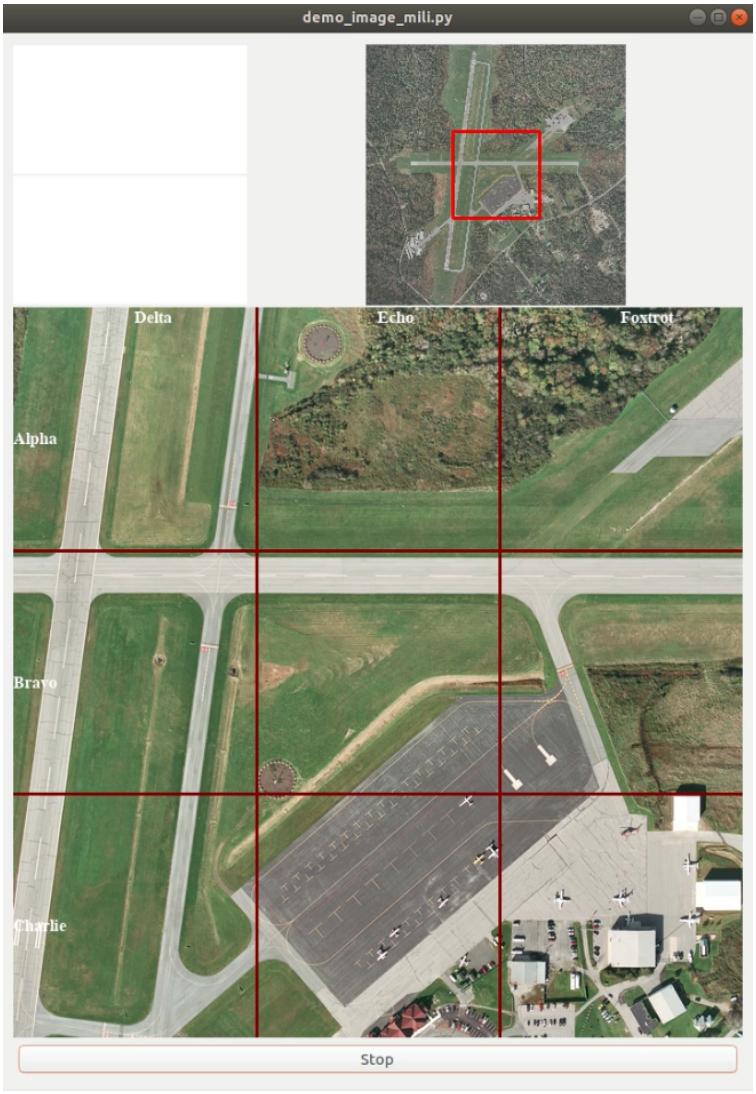

FIGURE 6.5 – Nouvelle image et localisation sur la carte

6.3.4 Annotation

La commande **annotation** permet à l'utilisateur de marquer des objets d'intérêts. Les objets à reconnaître sont à définir dans le modèle de langage du système de reconnaissance. Ici, on marque la position d'un avion avec la commande **annotation avion**.

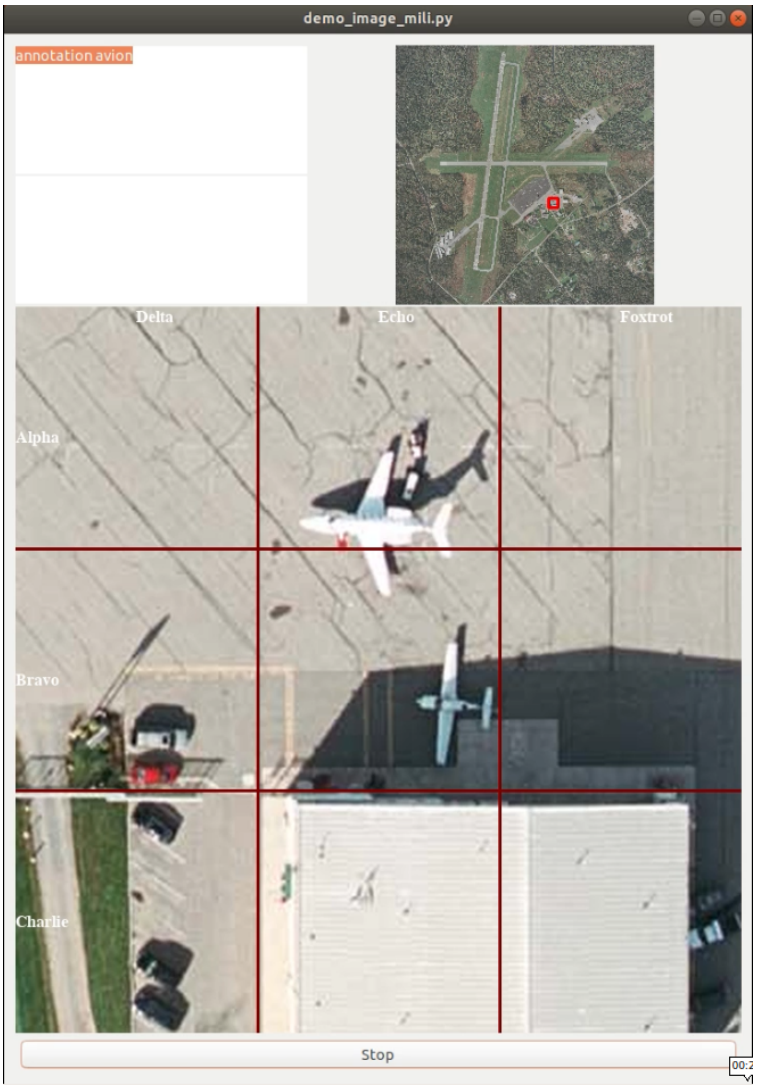

FIGURE 6.6 – Commande d'annotation

Une fois la commande reconnue, la cible est notée dans le second *buffer* et un marquage est placé sur la carte de localisation. La taille du marqueur est inversement proportionnelle au degré de zoom actuel.

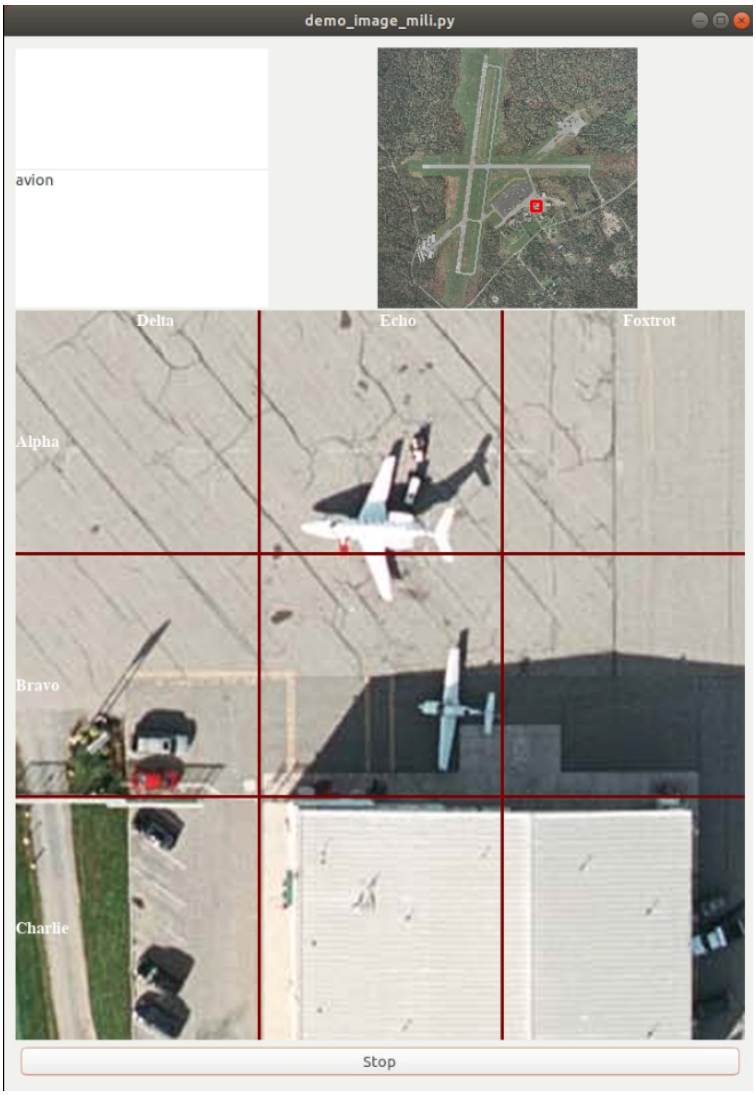

FIGURE 6.7 – Enregistrement et marquage sur la carte

6.3.5 Initialisation

Enfin, la commande **initialisation** permet à l'utilisateur de retourner au menu afin de se déplacer à nouveau et marquer d'autres cibles.

Chapitre 7

Conclusion et perspectives

Les connaissances et les compétences acquisent au cours de mon cursus à l'ENSTA Bretagne et durant ce stage m'ont permis de mener à bien ce projet. J'ai pu exercé pleinement le métier d'ingénieur dans l'appréhension, la compréhension et la résolution de problèmes techniques complexes. Toujours dans l'optique de développer une solution durable et facilement adaptable. De plus, un vrai travail de communication dans la restitution des résultats, tant écrite qu'orale, a été accompli.

Les résultats obtenus sont très satisfaisant compte tenu du délai imparti. Néanmoins, il est possible d'aller encore plus loin. De manière générale, l'accès à plus de données permettrait d'améliorer les performances à tous les niveaux (données audio pour le modèle acoustique et données textuelles pour le modèle de langage).
Par ailleurs, le modèle acoustique est améliorable par l'utilisation d'architecture de réseaux de neurones encore plus complexes (RNN, LSTM, Li-GRU...). Dont l'implémentation est simplifiée par l'utilisation de l'outil *Pytorch-Kaldi*. L'idée étant d'entraîner ces nouveaux modèles à partir des meilleurs alignements disponibles (modèle DNN Kaldi).

Le démonstrateur est également perfectible. Au lieu d'un modèle de langage équiprobable, une grammaire plus contraignante régissant la syntaxe de commande est possible : mot clé de commande – objet cible – description de la cible.
Une autre fonctionnalité intéressante est d'utiliser deux systèmes de reconnaissance en parallèle. Le premier pour reconnaître la commande vocale à partir d'un vocabulaire restreint et le second pour faire de l'annotation libre à partir d'un vocabulaire large, conciliant ainsi les bénéfices des deux systèmes.

Annexe A

Codes

```
  import   os


def clean_dir(directory):
    for root, directories, files in os.walk(directory):
        for file in files:
#            print(file)
            rel_dir = os.path.relpath(root, directory)
            rel_file = os.path.join(directory, rel_dir, file)
            os.remove(rel_file)
    return

clean_dir("./data_clean")

test_spk = ('FR091', 'FR092', 'FR093', 'FR094', 'FR095', 'FR096', 'FR097', 'FR098')
dev_spk = ('FR082', 'FR083', 'FR084', 'FR085', 'FR086', 'FR087', 'FR088', 'FR089')

dir_trl = "./trl/"
dir_wav = "./wav/"

list_trl = os.listdir(dir_trl)
list_wav = os.listdir(dir_wav)
data = list_wav.copy()



with open( "./data_clean/train/wav.scp", 'w') as train:
    with open( "./data_clean/test/wav.scp", 'w') as test:
        with open( "./data_clean/dev/wav.scp", 'w') as dev:
            for d in data:
                utt_id = d.split('.')[0]
                spk_id = utt_id[:5]
                wav_path = "/home/marc/Documents/BDD/Global_phone/wav/"+d
                if spk_id in test_spk:
                    test.write(utt_id+' '+wav_path+"\n")

                elif spk_id in dev_spk:
                    dev.write(utt_id+' '+wav_path+"\n")

                else:
                    train.write(utt_id+' '+wav_path+"\n")
```

```
42
43
44
45 trans_dict = {}
46 for trl in list_trl:
47     with open( dir_trl+trl, 'r', encoding = "ISO-8859-1") as f:
48         data = f.read()
49         data = data.replace('.','')
50         data = data.replace('?','')
51         data = data.replace(',','')
52         data = data.replace('!','')
53         data = data.replace('"','')
54         data = data.replace('  ','e')
55         data = data.replace('  ','e')
56         data = data.replace('  ','e')
57         data = data.replace('  ','a')
58         data = data.replace('  ','u')
59         data = data.replace('  ','u')
60         data = data.replace("'"," ")
61         data = data.replace('-',' ')
62         data = data.replace('--','')
63         data = data.replace('\n','')
64         data = data.lower()
65
66         data = data.split("; ")
67
68         line0 = data.pop(0) # ;SprecherID FR001
69         spk_id = line0.split()[1].upper()
70         trans = []
71
72         for d in data:
73             d = d.split(":")
74             trans.append(d)
75
76         trans_dict[spk_id] = trans
77
78
79
80 test_dict = {k:trans_dict[k] for k in test_spk if k in trans_dict}
81
82 dev_dict = {k:trans_dict[k] for k in dev_spk if k in trans_dict}
83
84 train_dict = trans_dict.copy()
85 keys = ('FR091', 'FR092', 'FR093', 'FR094', 'FR095', 'FR096', 'FR097', 'FR098', '
       FR082', 'FR083', 'FR084', 'FR085', 'FR086', 'FR087', 'FR088', 'FR089')
86 for key in keys:
87     if key in train_dict:
88         del train_dict[key]
89
90
91
92 with open( "./data_clean/test/text", 'w' ) as f:
93     with open( "./data_clean/test/utt2spk", 'w') as f2:
94         for key in test_dict.keys():
95             for data in test_dict[key]:
96                 utt = key+'_'+data[0]
97                 text = data[1]
98                 f.write(utt+" "+text+"\n")
99                 f2.write(utt+" "+key+"\n")
```

```
100
101 with open( "./data_clean/dev/text", 'w' ) as f:
102     with open( "./data_clean/dev/utt2spk", 'w') as f2:
103         for key in dev_dict.keys():
104             for data in dev_dict[key]:
105                 utt = key+'_'+data[0]
106                 text = data[1]
107                 f.write(utt+" "+text+"\n")
108                 f2.write(utt+" "+key+"\n")
109
110 with open( "./data_clean/train/text", 'w' ) as f:
111         with open( "./data_clean/train/utt2spk", 'w') as f2:
112             for key in train_dict.keys():
113                 for data in train_dict[key]:
114                     utt = key+'_'+data[0]
115                     text = data[1]
116                     f.write(utt+" "+text+"\n")
117                     f2.write(utt+" "+key+"\n")
```

Listing A.1 – Préparation des données GlobalPhone au format Kaldi

```
1  import math
2
3
4  def writeArpa(words):
5      N_words = len(words)
6      ps = math.log10(0.5)
7      pw = math.log10(1/(N_words*2))
8      lines = ['\n', '\\data\\\n', 'ngram 1=%d\n' % (N_words+2), '\n', '\\1-grams:\
       n']
9      lines.append('%f\t</s>\n' % (ps))
10     lines.append('-99\t<s>\n')
11     for w in words:
12         lines.append('%f\t%s\n' % (pw, w))
13     lines.append('\n')
14     lines.append('\\end\\\n')
15     return lines
16
17 def writeVocab(words):
18     lines = ['-pau-\n', '</s>\n', '<s>\n', '<unk>\n']
19     for w in words:
20         lines.append('%s\n' % (w))
21     return lines
22
23
24 if __name__ == "__main__":
25     #words = ["zero", "one", "two", "three", "four",
26     #         "five", "six", "seven", "eight", "nine"]
27     words = ["oui", "non", "retour", "ecrit", "zoom","annotation","arriere","
       initialisation",
28        "zero",  "un", "deux", "trois", "quatre", "cinq", "six", "sept", "huit", "
       neuf",
29        "alpha", "bravo", "charlie", "delta", "echo", "foxtrot",
30        "avion", "arbre", "batiment", "camion", "terrain"]
31
32     f = open("lm.arpa", "w")
33     lines = writeArpa(words)
34     for line in lines:
35         f.write(line)
36     f.close()
```

```
37
38 #     f = open("vocab-full.txt", "w")
39 #     lines = writeVocab(words)
40 #     for line in lines:
41 #         f.write(line)
42 #     f.close()
```

Listing A.2 – Génération du ficher arpa équiprobable

```
1  #!/bin/bash
2  . ./path.sh || exit 1
3  . ./cmd.sh || exit 1
4  nj=15
5  nj_decode=8
6  lm_order=3
7  exp_dir=exp
8  # Safety mechanism (possible running this script with modified arguments)
9  . utils/parse_options.sh || exit 1
10 [[ $# -ge 1 ]] && { echo "Wrong arguments!"; exit 1; }
11 # Removing previously created data (from last run.sh execution)
12 rm -rf $exp_dir mfcc data/train/spk2utt data/train/cmvn.scp data/train/feats.scp
       data/test/spk2utt data/test/cmvn.scp data/test/feats.scp data/dev/spk2utt data
       /dev/cmvn.scp data/dev/feats.scp data/local/lang data/lang data/local/tmp data
       /local/dict/lexiconp.txt
13
14 # Acoustic model parameters
15 #FR 3100  50000
16 #numLeavesTri1=1000
17 #numGaussTri1=30000
18 #numLeavesMLLT=1000
19 #numGaussMLLT=30000
20 #numLeavesSAT=1000
21 #numGaussSAT=30000
22
23 numLeavesTri1=3100
24 numGaussTri1=50000
25 numLeavesMLLT=3100
26 numGaussMLLT=50000
27 numLeavesSAT=3100
28 numGaussSAT=50000
29
30 echo ================================================================================
31 echo "                Data & Lexicon & Language Preparation                        "
32 echo ================================================================================
33
34
35 utils/utt2spk_to_spk2utt.pl data/train/utt2spk > data/train/spk2utt
36 utils/utt2spk_to_spk2utt.pl data/test/utt2spk > data/test/spk2utt
37 utils/utt2spk_to_spk2utt.pl data/dev/utt2spk > data/dev/spk2utt
38
39 echo
40 echo "===== PREPARING LANGUAGE DATA ====="
41 echo
42 utils/prepare_lang.sh data/local/dict "<UNK>" data/local/lang data/lang
43
44 echo
45 echo "===== LANGUAGE MODEL CREATION ====="
46 echo "===== MAKING lm.arpa ====="
47 echo
48
```

```
loc=`which ngram-count`;
if [ -z $loc ]; then
  if uname -a | grep 64 >/dev/null; then
    sdir=$KALDI_ROOT/tools/srilm/bin/i686-m64
  else
    sdir=$KALDI_ROOT/tools/srilm/bin/i686
  fi
  if [ -f $sdir/ngram-count ]; then
    echo "Using SRILM language modelling tool from $sdir"
    export PATH=$PATH:$sdir
  else
    echo "SRILM toolkit is probably not installed. Instructions: tools/
   install_srilm.sh"
    exit 1
  fi
fi
local=data/local
mkdir $local/tmp
ngram-count -order $lm_order -write-vocab $local/tmp/vocab-full.txt -wbdiscount -
   text data/text.txt -lm $local/tmp/lm.arpa


echo
echo "===== MAKING G.fst ====="
echo
lang=data/lang
arpa2fst --disambig-symbol=#0 --read-symbol-table=$lang/words.txt $local/tmp/lm.
   arpa > $lang/G.fst


echo ============================================================================
echo "         MFCC Feature Extration & CMVN for Training and Test set          "
echo ============================================================================

echo
echo "===== FEATURES EXTRACTION ====="
echo
# Making feats.scp files
mfccdir=mfcc
utils/validate_data_dir.sh data/train
utils/validate_data_dir.sh data/test
utils/validate_data_dir.sh data/dev
# script for checking if prepared data is all right
utils/fix_data_dir.sh data/train
utils/fix_data_dir.sh data/test
utils/fix_data_dir.sh data/dev
# tool for data sorting if something goes wrong above
steps/make_mfcc.sh --mfcc-config conf/mfcc.conf --nj $nj --cmd "$train_cmd" data/
   train $exp_dir/make_mfcc/train $mfccdir
steps/make_mfcc.sh --mfcc-config conf/mfcc.conf --nj $nj_decode --cmd "$train_cmd
   " data/test $exp_dir/make_mfcc/test $mfccdir
steps/make_mfcc.sh --mfcc-config conf/mfcc.conf --nj $nj_decode --cmd "$train_cmd
   " data/dev $exp_dir/make_mfcc/dev $mfccdir
# Making cmvn.scp files
steps/compute_cmvn_stats.sh data/train $exp_dir/make_mfcc/train $mfccdir
steps/compute_cmvn_stats.sh data/test $exp_dir/make_mfcc/test $mfccdir
steps/compute_cmvn_stats.sh data/dev $exp_dir/make_mfcc/dev $mfccdir


```

```
102
103
104 echo ============================================================================
105 echo "                      MonoPhone Training & Decoding                       "
106 echo ============================================================================
107 #training
108 steps/train_mono.sh --nj $nj --cmd "$train_cmd" data/train data/lang $exp_dir/
    mono || exit 1
109
110 #decoding
111 utils/mkgraph.sh --mono data/lang $exp_dir/mono $exp_dir/mono/graph || exit 1
112
113 steps/decode.sh --config conf/decode.config --nj $nj_decode --cmd "$decode_cmd" \
114  $exp_dir/mono/graph data/test $exp_dir/mono/decode_test
115 steps/decode.sh --config conf/decode.config --nj $nj_decode --cmd "$decode_cmd" \
116  $exp_dir/mono/graph data/dev $exp_dir/mono/decode_dev
117
118 for x in $exp_dir/mono/decode_test; do
119 [ -d $x ] && grep WER $x/wer_* | \
120     utils/best_wer.sh > wer/mono_wer.txt;
121 done
122
123 #alignment
124 steps/align_si.sh --nj $nj --cmd "$train_cmd" data/train data/lang $exp_dir/mono
    $exp_dir/mono_ali || exit 1
125 steps/align_si.sh --nj $nj_decode --cmd "$train_cmd" data/test data/lang $exp_dir
    /mono $exp_dir/mono_ali_test || exit 1
126 steps/align_si.sh --nj $nj_decode --cmd "$train_cmd" data/dev data/lang $exp_dir/
    mono $exp_dir/mono_ali_dev || exit 1
127
128
129
130
131 echo ============================================================================
132 echo "           tri1 : Deltas + Delta-Deltas Training & Decoding               "
133 echo ============================================================================
134 #training
135 steps/train_deltas.sh --cmd "$train_cmd" $numLeavesTri1 $numGaussTri1 data/train
    data/lang $exp_dir/mono_ali $exp_dir/tri1 || exit 1
136
137 #decoding
138 utils/mkgraph.sh data/lang $exp_dir/tri1 $exp_dir/tri1/graph || exit 1
139
140 steps/decode.sh --config conf/decode.config --nj $nj_decode --cmd "$decode_cmd" \
141  $exp_dir/tri1/graph data/test $exp_dir/tri1/decode_test
142 steps/decode.sh --config conf/decode.config --nj $nj_decode --cmd "$decode_cmd" \
143  $exp_dir/tri1/graph data/dev $exp_dir/tri1/decode_dev
144
145 for x in $exp_dir/tri1/decode_test; do
146 [ -d $x ] && grep WER $x/wer_* | \
147     utils/best_wer.sh > wer/tri1_wer.txt;
148 done
149
150 #alignment
151 steps/align_si.sh --nj $nj --cmd "$train_cmd" --use-graphs true \
152  data/train data/lang $exp_dir/tri1 $exp_dir/tri1_ali
153 steps/align_si.sh --nj $nj_decode --cmd "$train_cmd" \
154  data/test data/lang $exp_dir/tri1 $exp_dir/tri1_ali_test
155 steps/align_si.sh --nj $nj_decode --cmd "$train_cmd" \
```

```
 data/dev data/lang $exp_dir/tri1 $exp_dir/tri1_ali_dev



echo ============================================================================
echo "                 tri2b : LDA + MLLT Training & Decoding                   "
echo ============================================================================
# train and decode tri2b [LDA+MLLT]
steps/train_lda_mllt.sh --cmd "$train_cmd" \
  --splice-opts "--left-context=3 --right-context=3" \
 $numLeavesMLLT $numGaussMLLT data/train data/lang $exp_dir/tri1_ali $exp_dir/
    tri2b
utils/mkgraph.sh data/lang $exp_dir/tri2b $exp_dir/tri2b/graph

steps/decode.sh --config conf/decode.config --nj $nj_decode --cmd "$decode_cmd" \
   $exp_dir/tri2b/graph data/test $exp_dir/tri2b/decode_test
steps/decode.sh --config conf/decode.config --nj $nj_decode --cmd "$decode_cmd" \
   $exp_dir/tri2b/graph data/dev $exp_dir/tri2b/decode_dev

for x in $exp_dir/tri2b/decode_test; do
[ -d $x ] && grep WER $x/wer_* | \
    utils/best_wer.sh > wer/tri2b_wer.txt;
done

# Align all data with LDA+MLLT system (tri2b)
steps/align_si.sh --nj $nj --cmd "$train_cmd" --use-graphs true \
   data/train data/lang $exp_dir/tri2b $exp_dir/tri2b_ali
steps/align_si.sh --nj $nj_decode --cmd "$train_cmd" \
   data/test data/lang $exp_dir/tri2b $exp_dir/tri2b_ali_test
steps/align_si.sh --nj $nj_decode --cmd "$train_cmd" \
   data/dev data/lang $exp_dir/tri2b $exp_dir/tri2b_ali_dev



echo ============================================================================
echo "              tri3b : LDA + MLLT + SAT Training & Decoding                "
echo ============================================================================
## Do LDA+MLLT+SAT, and decode.
steps/train_sat.sh $numLeavesSAT $numGaussSAT data/train data/lang $exp_dir/
    tri2b_ali $exp_dir/tri3b
utils/mkgraph.sh data/lang $exp_dir/tri3b $exp_dir/tri3b/graph

steps/decode_fmllr.sh --config conf/decode.config --nj $nj_decode --cmd "
    $decode_cmd" \
  $exp_dir/tri3b/graph data/test $exp_dir/tri3b/decode_test
steps/decode_fmllr.sh --config conf/decode.config --nj $nj_decode --cmd "
    $decode_cmd" \
  $exp_dir/tri3b/graph data/dev $exp_dir/tri3b/decode_dev

for x in $exp_dir/tri3b/decode_test; do
[ -d $x ] && grep WER $x/wer_* | \
    utils/best_wer.sh > wer/tri3b_wer.txt;
done

# Align all data with LDA+MLLT+SAT system (tri3b)
steps/align_fmllr.sh --nj $nj --cmd "$train_cmd" --use-graphs true \
  data/train data/lang $exp_dir/tri3b $exp_dir/tri3b_ali
```

```
steps/align_fmllr.sh --nj $nj_decode --cmd "$train_cmd" \
  data/test data/lang $exp_dir/tri3b $exp_dir/tri3b_ali_test
steps/align_fmllr.sh --nj $nj_decode --cmd "$train_cmd" \
  data/dev data/lang $exp_dir/tri3b $exp_dir/tri3b_ali_dev



echo
echo "===== run.sh script is finished ====="
echo
```

Listing A.3 – Recette Kaldi

Annexe B

Documents

ENSTA Bretagne

Autorisation de diffusion d'un rapport de PFE
Final Year Project Report Authorization

A remplir par **l'ingénieur pilote et l'élève**
To be completed by the project supervisor and the student

La diffusion du rapport se limite aux lecteurs de la médiathèque de l'ENSTA Bretagne à des fins pédagogiques.
The report will only be made available to ENSTA Bretagne Multimedia Library Readers for pedagogical ends.

Ingénieur pilote *(Project supervisor)* **:** Marc MICHAU

Etablissement *(Firm or institution)* **:** Thales SIX GTS

Etudiant *(Student)* **:** Cyril COTSAFTIS

Promotion *(Class year)* **:** CI2020

Option de 3ème année *(3rd year option)* **:** ROB

Titre du rapport *(Title of report)* : Mise en oeuvre de techniques d'intelligence artificielle pour la reconnaissance vocale

- Autorisez-vous :
  Do you authorize :

  - la consultation du rapport à la médiathèque de l'ENSTA Bretagne.
    consultation of the report in ENSTA Bretagne Multimedia Library.

    ☑ Oui □ Non

  - le prêt du rapport.
    loan of the report.

    ☑ Oui □ Non

  - la diffusion du rapport en texte intégral dans le catalogue de la médiathèque.
    a complete copy of the report to be made available in the Multimedia Library Catalogue.

    ☑ Oui □ Non

Commentaires éventuels *(Any further comments)* :

Date : 07/12/2020

Fonction et signature de l'ingénieur pilote :
Title and signature of the project supervisor:

Date : 07/12/2020

Signature de l'étudiant :
Signature of the student:

Merci de retourner ce document complété à la médiathèque avant votre soutenance par mail à : mediatheque@ensta-bretagne.fr OU de le glisser dans un exemplaire du rapport.
Please return the completed document to the Multimedia Library by mail at mediatheque@ensta-bretagne.fr before your oral presentation OR insert it in a copy of the report.

FIGURE B.1 – Autorisation de diffusion du rapport

ENSTA Bretagne	**FICHE D'APPRECIATION DE STAGE** **A renseigner et à viser par l'encadrant entreprise puis à téléverser dans aurion – Rubrique Mise à jour de PFE – au format pdf**

Organisme	Thales SIX GTS France
Dates du stage	du 01/07/2020 au 11/12/2020
NOM, Prénom du stagiaire	COTSAFTIS Cyril

	Cocher les cases appropriées					
	F (échec)	E (insuffisant)	D (passable)	C (assez bien à bien)	B (bien à très bien)	A (remarquable)
Critères d'intégration – Savoir être						
Adaptabilité						X
Disponibilité						X
Culture de l'entreprise						X
Puissance de travail						X
Qualité d'expression						X
Conduite du projet						
Identification des tâches					X	
Organisation/répartition des tâches dans le temps						X
Respect des délais des livrables demandés						X
Force de proposition						X
Éventuellement : travail en équipe						
Rapport de stage						
Forme (présentation, style…)					X	
Fond (exactitude)						X
Exploitabilité par l'organisme						X

Appréciation générale

Tout au long de son stage, Cyril a montré des capacités remarquables de compréhension de sujets complexes et de prise en main d'outils logiciels variés. Les résultats qu'il a obtenus, ainsi que le démonstrateur qu'il a mis en place sont des livrables de qualité, totalement en accord avec les objectifs de son stage. Très autonome, il a fait preuve d'une grande adaptabilité dans son travail.

Si vous disposiez d'un poste correspondant au profil du stagiaire, souhaiteriez-vous l'embaucher ?	☑ OUI □ NON

NOM, Prénom de l'encadrant entreprise : MICHAU Marc **Date :** 07/12/2020

Fonction : Ingénieur R&D Audio **Signature :**

FIGURE B.2 – Fiche d'appréciation de stage

Bibliographie

[1] Decode kaldi. https://kaldi-asr.org/doc/graph.html.

[2] guide-mel-frequency-cepstral-coefficients-mfccs. http://practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/.

[3] Programme contact. https://www.thalesgroup.com/fr/programme-contact.

[4] Speechbrain. https://speechbrain.github.io/.

[5] Andrew L. Maas and Peng Qi and Ziang Xie and Awni Y. Hannun and Christopher T. Lengerich and Daniel Jurafsky and Andrew Y. Ng. Building dnn acoustic models for large vocabulary speech recognition. *Computer Speech Language*, 41 :195 – 213, 2017.

[6] Benesty, Jacob and Sondhi, M. Mohan and Huang, Yiteng (Arden). *Springer Handbook of Speech Processing*. Springer-Verlag, Berlin, Heidelberg, 2007.

[7] Daniel Povey and Xiaohui Zhang and Sanjeev Khudanpur. Parallel training of dnns with natural gradient and parameter averaging, 2015.

[8] Dong Yu and L. Deng. *Automatic Speech Recognition : A Deep Learning Approach*. 2014.

[9] Joseph Mariani (sous la direction de). *Reconnaissance de la parole - Traitement automatique du langage parlé 2*. Hermès Science publications, 2002.

[10] Julia Hirschberg. Chapter 9 : Automatic speech recognition. http://www.cs.columbia.edu/~julia/courses/CS6998-2019/%5b09%5d%20Automatic%20Speech%20Recognition.pdf, 2019.

[11] Mohri, Mehryar and Pereira, Fernando and Riley, Michael. Weighted finite-state transducers in speech recognition. *Comput. Speech Lang.*, 16(1) :69–88, 2002.

[12] Povey, Daniel and Ghoshal, Arnab and Boulianne, Gilles and Burget, Lukas and Glembek, Ondrej and Goel, Nagendra and Hannemann, Mirko and Motlicek, Petr and Qian, Yanmin and Schwarz, Petr and Silovsky, Jan and Stemmer, Georg and Vesely, Karel. The kaldi speech recognition toolkit. In *IEEE 2011 Workshop on Automatic Speech Recognition and Understanding*. IEEE Signal Processing Society, Dec. 2011. IEEE Catalog No. : CFP11SRW-USB.

[13] Rath, Shakti P. and Povey, Daniel and Veselý, Karel and Cernocký, Jan. Improved feature processing for deep neural networks. In F. Bimbot, C. Cerisara, C. Fougeron, G. Gravier, L. Lamel, F. Pellegrino, and P. Perrier, editors, *INTERSPEECH*, pages 109–113. ISCA, 2013.

[14] Sihem Romdhani. Implementation of dnn-hmm acoustic models for phoneme recognition. 2014.

[15] Victor Rosillo Gil. Automatic speech recognition with kaldi toolkit. pages 9 – 11.

[16] X. Zhang and J. Trmal and D. Povey and S. Khudanpu. Improving deep neural network acoustic models using generalized maxout networks. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 215–219, 2014.

[17] Xuedong Huang, Alex Acero, Hsiao-Wuen Hon. *Spoken Language Processing*. Prentice-Hall PTR, 2001.