Mise au point d'un système de reconnaissance de la parole de l'enseignant

Jérôme Aragona

Mise au point d'un système de reconnaissance de la parole de l'enseignant

Jérôme ARAGONA

Sous la direction de :

Solange ROSSATO et Solène EVAIN

Professeur référent :

Thomas LEBARBÉ

Laboratoire d'Informatique de Grenoble (LIG)

UFR LLASIC

Département I3L

Rapport de stage de master 2 Sciences du langage parcours industries de la langue - 20 crédits

Année universitaire 2022-2023

Mise au point d'un système de reconnaissance de la parole de l'enseignant

Jérôme ARAGONA

Sous la direction de :

Solange ROSSATO et Solène EVAIN

Professeur référent :

Thomas LEBARBÉ

Laboratoire d'Informatique de Grenoble (LIG)

UFR LLASIC

Département I3L

Rapport de stage de master 2 Sciences du langage parcours industries de la langue - 20 crédits

Année universitaire 2022-2023

Remerciements :

Je tiens à remercier les personnes suivantes qui ont contribué au bon déroulement de mon stage et qui m'ont aidé lors de la rédaction de ce mémoire.

Madame Solange ROSSATO, sans qui je n'aurais peut-être pas pu faire ce stage, pour ses conseils, sa patience et son soutien durant ces six derniers mois. En particulier pour ses conseils d'organisation, conseils qui, je l'avoue, n'ont pas toujours été suivis, mais qui m'ont grandement aidé à reprendre pied dans ce stage à un moment où je commençais à m'enliser.

Madame Solène EVAIN, sans qui, je pense, je serais resté bloqué sur des problèmes pendant vraiment très longtemps, pour sa disponibilité, sa patience et sa grande aide sur de nombreux sujets. Je la remercie aussi grandement pour ses multiples remarques sur ce mémoire, ces dernières m'ayant énormément aidé.

Mes ami.e.s, également en stage au labo d'autres bureaux, Elena, Chloé, Romain et Sophie, auprès de qui j'ai pu obtenir du réconfort dans les moments difficile et oublier un peu les problèmes du quotidien. Mais aussi et surtout rire et échanger sur nos stages respectifs durant des pauses repas + jeux de sociétés endiablées.

Adrien PUPIERA, pour son temps passé avec moi à essayer de faire fonctionner mes modèles récalcitrant même si ces derniers ne fonctionnent toujours pas.

Mes collègues et amis du bureau 323, Marko, Soline, Xingyu, Eric et Sannara pour l'aide donnée et surtout la joie ambiante de ce bureau, les rires et la bonne humeur.

Merci également à mes scouts du groupe des Pionniers - Caravelles de Meylan dont je suis chef pour cette super semaine de radeaux sur l'Isle, juste avant la dernière ligne droite de ce stage et de ce mémoire. Cette dernière m'a permis de me vider la tête et d'arriver sans stress (mais exténué) pour la partie la plus stressante et éreintante de ce stage.

Université Grenoble Alpes

DÉCLARATION ANTI-PLAGIAT

1. Ce travail est le fruit d'un travail personnel et constitue un document original.
2. Je sais que prétendre être l'auteur d'un travail écrit par une autre personne est une pratique sévèrement sanctionnée par la loi.
3. Personne d'autre que moi n'a le droit de faire valoir ce travail, en totalité ou en partie, comme le sien.
4. Les propos repris mot à mot à d'autres auteurs figurent entre guillemets (citations).
5. Les écrits sur lesquels je m'appuie dans ce mémoire sont systématiquement référencés selon un système de renvoi bibliographique clair et précis.

PRENOM : Jérôme

NOM : ARAGONA

DATE : 31/08/2023

Table des matières

Introduction

Dans une optique d'apprentissage et de développement de mes connaissances grâce à une expérience concrète, j'ai saisi l'opportunité d'effectuer mon stage de fin d'études dans le domaine de la reconnaissance automatique de la parole. Ce domaine m'intéresse particulièrement et je souhaite y développer une expertise professionnelle en vue d'une potentielle carrière dans la reconnaissance automatique de la parole ou du moins dans le traitement de l'oral. Aussi, ce stage constitue une précieuse opportunité d'acquérir une première expérience concrète sur un projet de recherche appliquée en situation réelle. Il me permettra également d'approfondir mes connaissances théoriques dans le domaine de la reconnaissance de parole en y ajoutant des connaissances pratiques et en apprenant davantage sur les domaines liés au monde du travail et de la recherche.
Mon intérêt pour le traitement automatique des langues, et plus spécifiquement pour la reconnaissance automatique de la parole, m'a conduit à postuler au stage proposé par les laboratoires du LIG et du LIDILEM pour mon stage de validation de Master deux. Cette opportunité correspond à la fois à mon parcours universitaire et à mon projet professionnel. De plus, ce dernier me permettra de découvrir le milieu de la recherche en laboratoire ainsi que le fonctionnement d'une équipe de recherche me permettant alors d'en connaître plus sur ce milieu et ainsi de savoir si je pourrais continuer à travailler dans la recherche, si j'apprécie ce milieu ou si je me dirige plutôt vers un milieu professionnel en entreprise, voire vers un autre domaine que le traitement automatique des langues.

La reconnaissance automatique de la parole est un domaine de recherche actif depuis plusieurs décennies. Les premiers systèmes, dans les années 1950, ne permettaient de reconnaître que quelques mots isolés avec un vocabulaire limité. Ce dernier est aujourd'hui un domaine en plein essor, notamment grâce aux avancées récentes en deep learning. Cependant, certains contextes d'enregistrement, tel que, comme nous le verrons, la parole adressée aux jeunes enfants en milieu scolaire, représentent encore un défi important.

Dans le cadre de l'enseignement, la reconnaissance automatique de la parole des professeurs présente un intérêt certain. Elle pourrait, entre autres choses, permettre des applications telles que la retranscription automatique du cours ou l'analyse du discours pédagogique et l'évolution de ce dernier sur d'autres corpus n'ayant pas été transcrits manuellement, voire permettre la transcription des enregistrements de paroles d'enseignants du projet DyLNet n'ayant pas été transcrit manuellement ce qui serait un vrai gain de temps (il reste 3700 heures d'enregistrement d'enfants et

d'adulte non transcrites). Néanmoins, la salle de classe est un environnement acoustique difficile : bruits ambiants, mauvaise qualité sonore, parole d'enfants sur les enregistrements d'adultes et inversement, etc. De plus, le vocabulaire employé peut être spécifique au domaine et au niveau scolaire. Celui-ci peut aussi varier d'un enseignant à l'autre et du moment de l'enregistrement (la parole de l'enseignant envers l'élève peut être différente de la parole de l'enseignant envers un autre adulte).
Ma mission pour ce stage est de mettre au point un système de reconnaissance automatique de la parole adapté à la parole de l'enseignant en classe de maternelle, dans le cadre du projet DyLNet. Ce projet vise à étudier l'influence des interactions sociales sur le développement du langage oral des enfants à l'école maternelle, ainsi, l'étude du discours de l'enseignant et de sa possible évolution au cours des années et en fonction du niveau des élèves pourrait avoir un intérêt réel dans l'étude du développement du langage oral des enfants à l'école maternelle. Mon objectif principal est donc de développer un système performant et robuste de reconnaissance automatique de la parole de l'enseignant, permettant de transcrire automatiquement les enregistrements de parole des enseignants s'adressant aux élèves ou non, qui ont été enregistrés lors de la collecte de données du projet DyLNet tout en s'adaptant aux différents locuteurs et conditions d'enregistrement. Ce stage représente une précieuse opportunité du fait de sa contribution à un projet de recherche appliquée.

Le présent rapport de stage est structuré en plusieurs parties qui me permettent de présenter ma recherche sur le développement d'un système de reconnaissance de la parole de l'enseignant. Dans un premier temps, je présente le contexte général du projet en décrivant le laboratoire d'Informatique de Grenoble (LIG) ainsi que l'équipe de recherche avec laquelle j'ai travaillé, en mettant en évidence le projet DyLNet. Ensuite, je détaille le matériel et la méthode utilisés, en présentant l'environnement informatique, les données de DyLNet et l'outil Speechbrain qui ont été au cœur de mes expérimentations. Dans la troisième partie, je présente les résultats finaux obtenus et engage une discussion sur ces résultats, en mettant en évidence les performances du système de reconnaissance de la parole que j'ai développée. Enfin, je conclus ce rapport en récapitulant les principaux résultats et en discutant des perspectives d'amélioration et de mon ressenti par rapport à ce stage.

1 Contexte général

1.1 Présentation du laboratoire et de l'équipe

J'ai effectué mon stage sur le campus de l'université Grenoble Alpes à Saint-Martin-d'Hères au laboratoire d'informatique de Grenoble (LIG). Le LIG est un laboratoire de recherche informatique créé en 2007, qui regroupe près de quatre-cent-cinquante chercheurs, doctorants, enseignants-chercheurs et stagiaires répartis sur trois sites différents : le campus de l'université Grenoble Alpes, le campus Minatec à Grenoble et l'INRIA à Montbonnot. Les chercheurs du LIG travaillent au sein de vingt-deux équipes de recherche, avec le soutien du personnel dédié à la recherche.

Le LIG se concentre sur cinq axes de recherche majeurs. Le premier axe concerne le traitement de données et de connaissances à grande échelle, tandis que le deuxième axe se concentre sur le génie des logiciels et des systèmes d'informations. Le troisième axe concerne les méthodes formelles et les modèles de langage, tandis que le quatrième axe se penche sur les systèmes interactifs et cognitifs. Enfin, le cinquième axe porte sur les systèmes répartis, le calcul parallèle et les réseaux de systèmes.

Pour ma part, j'ai réalisé mon stage au sein de l'équipe Getalp, qui fait partie de l'axe "méthodes formelles et modèles de langage" et est basée sur le campus de l'université Grenoble Alpes. L'équipe Getalp, abréviation de Groupe d'Étude en Traduction Automatique/Traitement Automatisé des Langues et de la Parole, a été créée en 2007 en même temps que le LIG. Ses chercheurs se consacrent à la traduction automatique et au traitement automatique du langage.

Durant mon stage, j'ai été encadré par mesdames Solange ROSSATO et Solène Evain, qui ont été mes tutrices et m'ont apporté leur soutien technique tout au long de mon projet.

FIGURE 1 – Logo du LIG

1.2 Projet DyLNet

1.2.1 Présentation du projet

Le projet DyLNet (Dynamics of Language and sociability at preschool) est une étude à grande échelle qui vise à observer et étudier les relations entre la socialisation des enfants et l'apprentissage du langage oral pendant la période de l'école maternelle. Coordonné par Aurélie Nardy et financé par l'ANR (Agence Nationale de la Recherche), ce projet a été mené entre 2016 et 2021 en partenariat avec l'Inria DANTE (École Normale Supérieure de Lyon) Bouchet et al. [2017].

FIGURE 2 – Logo du projet DyLNet

L'objectif premier de DyLNet est d'étudier les interactions entre les enfants et le personnel enseignant (enseignants et assistants de classe) pour déterminer si la fréquence de ces interactions a un impact sur le développement du langage des enfants. Pour ce faire, le projet a suivi tous les enfants (200 enfants au total) et le personnel enseignant d'une école maternelle socialement mixte pendant trois ans.Bouchet et al. [2017]
Le projet DyLNet adopte une approche multidisciplinaire, combinant des travaux dans les domaines de l'acquisition du langage et la sociolinguistique. Quatre types de données ont été collectés au cours du projet : des données transactionnelles et vocales sur les interactions sociales et orales, des données sociodémographiques, des données sur les compétences linguistiques des enfants et enfin des données sur les réseaux de relations sociales des enfants.Bouchet et al. [2017]
Le but de cette étude est de mieux comprendre comment les relations sociales influencent l'apprentissage du langage oral chez les enfants en bas âge et d'identifier les facteurs clés qui contribuent à leur développement linguistique. En outre, le projet DyLNet fournira à la communauté scientifique une base de données indiquant les relations entre les fréquences d'interaction enregistrées et les compétences linguistiques

des enfants. Afin de collecter les données, les enfants ainsi que le personnel enseignant ont porté des microphones à détection de parole automatique, enregistrant chaque fois que le porteur parlait.

1.2.2 Mission du stage

La principale mission de ce stage est de proposer un système de reconnaissance automatique de la parole adapté aux enseignants. Pour atteindre cet objectif, le stage se concentrera donc sur la mise en place de ce système de reconnaissance de la parole spécifiquement conçu pour les conditions d'enregistrement, variables selon l'activité en cours (en cours, en récréation, etc . Cf. 2.2), où les adultes communiquent avec de très jeunes enfants d'école maternelle. La demande pour ce stage comprend la réalisation d'expérimentations avec plusieurs types d'entraînement d'un système de reconnaissance automatique de parole afin d'identifier le modèle le plus performant à utiliser. De plus, il m'a été demandé d'analyser les taux d'erreurs en fonction du locuteur et de l'activité en cours, dans le but d'étudier les variations de performance liées à ces deux variables. Pour la bonne réalisation de cette mission, "j'ai à ma disposition 44h d'enregistrements de 14 enseignants, ce qui représente 5.5% des 800 heures de données déjà transcrites sur les 4500h que compte le projet.

1.3 La reconnaissance automatique de parole

La reconnaissance automatique de parole, également appelée transcription automatique de parole, est une technologie informatique qui permet à une machine de transcrire de la parole humaine en texte. Les systèmes captent les sons produits par la voix humaine à l'aide d'un microphone, puis analysent ces sons vocaux pour les transformer en mots écrits. Cette technologie peut notamment être utilisée pour des applications de dictée vocale, pour la commande vocale, pour la transcription automatisée ou encore pour l'interaction verbale avec des assistants virtuels (Bell [2020]). Elle représente un domaine de recherche en intelligence artificielle qui vise à doter les machines d'une capacité à transcrire de la parole similaire à celle des humains.

les système de reconnaissance automatique de parole (ASR) peut être classé en plusieurs grandes catégories : les systèmes basés sur les modèles de Markov cachés (HMM) et les systèmes end-to-end (E2E) (Deleglise and Lailler [2020]). Les systèmes HMM/DNN sont des approches hybrides qui combinent les modèles de Markov ca-

chés avec des réseaux de neurones pour modéliser les signaux audio et les phonèmes, ces derniers fonctionnent grâce à trois blocs : le modèle acoustique (qui peut être HMM-DNN ou HMM-GMM), le modèle de langue et le lexique. Les systèmes End-to-end, en revanche, sont basés sur des modèles neuronaux qui apprennent directement la relation entre le signal audio et la transcription textuelle sans utiliser de connaissances phonétiques ou lexicales préalables. Le modèle acoustique est un élément clé d'un système ASR. Il représente la relation entre un signal audio et les phonèmes ou autres unités linguistiques qui composent la parole. Ils sont entraînés à partir d'un ensemble d'enregistrements audio et de leurs transcriptions correspondantes pour créer des représentations statistiques des sons qui composent chaque mot. Le modèle de langage est un autre composant essentiel d'un système ASR. Il est utilisé pour évaluer la probabilité des séquences de mots et aider le système à déterminer la séquence de mots la plus probable. Les modèles de langage peuvent être basés sur des approches statistiques traditionnelles, telles que les modèles de n-grammes (Jurafsky and Martin [2022]), ou sur des modèles neuronaux plus récents, tels que les réseaux de neurones récurrents (RNN), les réseaux de neurones profonds (DNN) et les Transformers. Les systèmes end-to-end quant à eux n'utilisent pas de lexiques et les modèles de langues appris sur des données textuelles sont optionnels.

Sur le papier, ces systèmes seraient capables d'avoir de bonnes performances sur les voix d'enseignants, car ces derniers se doivent d'avoir une parole claire et articulée, surtout dans le cadre de l'école maternelle. Cependant, certains accents et particularités de prononciation, notamment lié aux niveaux dans lequel enseigne le ou la professeure, peuvent poser problème et diminuer la précision. Un entraînement spécifique sur des voix d'enseignants, exerçant au niveau souhaité (ici en école maternelle), améliore significativement les résultats (NGUYEN [2022]).

La mise au point d'un tel système adapté aux enseignants en école maternelle présente des défis particuliers. En effet, le bruit ambiant et les interruptions fréquentes dans une salle de classe maternelle peuvent dégrader la qualité du signal vocal (Gelin [2022]). Ainsi, l'entraînement d'un modèle de reconnaissance vocale spécifique au langage enseignant-élève en maternelle, robuste au bruit ambiant des salles de classe, et adapté à la situation d'interaction des enseignants, est primordiale pour obtenir des performances satisfaisantes. Pour ce faire, nous avons à notre disposition les enregistrements et transcriptions de parole d'enseignants dans le corpus d'enregistrement du projet DyLNet.

2 Matériel et méthode

2.1 Environnement informatique

Dans un premier temps, il a été fondamental d'apprendre et de s'habituer à travailler sous un environnement Linux. Ce dernier, bien que peu différent de l'environnement Windows que j'avais l'habitude d'utiliser, a nécessité un temps d'adaptation, notamment pour les actions les plus « basiques » tel que l'utilisation du logiciel de traitement de texte « Vim », la manipulation des fichiers ou encore l'utilisation d'environnements virtuels « Conda ». De plus, afin de récupérer, stocker et utiliser les fichiers nécessaires aux différentes expérimentations (notamment les fichiers de DyLNet) l'accès aux serveurs et l'utilisation de ces derniers étant impératif. Il a alors fallu apprendre à les utiliser, et savoir quel serveur utiliser pour quelle utilité, notamment pour les serveurs GPU [1] dont l'accès se fait de façons spécifiques, chose qui m'a posé de nombreux problèmes au début. Ces derniers ont aussi, comme pour les serveurs « classiques », des puissances de calcul différentes. Ainsi, le serveur choisi dépend de ce que l'on souhaite faire (si le programme nécessite une puissance de calcul élevée ou non), mais aussi, et surtout, du nombre de GPU disponibles sur le serveur. En effet, ces derniers sont limités et leur accès est géré par un ordonnanceur permettant l'accès direct ou l'ajout en file d'attente sur les serveurs « lig-gpu », dans le cas où les GPU ne sont pas disponibles ou en cas de demandes trop élevées (trop de GPU ou trop de temps demandé à la fois) pour l'accès au nombre demandé de GPU sur un serveur. La demande d'accès à ces GPU se fait via une commande spécifique. Par exemple, la commande présentée plus bas (2.1) permet de lancer le script « run-train-with-wav2vec2.sh » sur deux GPU sur le serveur « lig-gpu9 », pendant une durée de 90 heures. Ce serveur est l'un des serveurs dont les GPU permettent la plus grande puissance de calcul puisqu'il a quatre GPU de 48 gigaoctets chacun. Ce dernier est celui que j'ai le plus utilisé du fait de sa puissance.

oarsub -l /host=1/gpu=2, walltime=90 :00 :00 -p " host='lig-gpu9.imag.fr'" './run-train-with-wav2vec2.sh'

Exemple de commande de soumission d'un script sur un serveur OAR (ici lig-gpu9)

1. un GPU est un processeur graphique permettant le calcul

Serveur	GPU	CPU	RAM
lig-gpu1	4x NVIDIA GTX 1080 Ti 11Go	2x Intel Xeon E5-2620 v4 @ 2.10GHz 8C/16T	256Go
lig-gpu2	4x NVIDIA GTX 1080 Ti 11Go	2x Intel Xeon E5-2620 v4 @ 2.10GHz 8C/16T	256Go
lig-gpu3	4x NVIDIA GTX 1080 Ti 11Go	2x Intel Xeon E5-2620 v4 @ 2.10GHz 8C/16T	256Go
lig-gpu4	4x NVIDIA GTX 1080 Ti 11Go	2x Intel Xeon Silver 4110 @ 2.10GHz 8C/16T	256Go
lig-gpu5	2x NVIDIA GTX 1080 Ti 11Go 2x NVIDIA TITAN X (Pascal) 12Go	2x Intel Xeon Silver 4110 @ 2.10GHz 8C/16T	512Go
lig-gpu6	4x NVIDIA RTX 2080 Ti 11Go	Intel(R) Xeon(R) Silver 4210 CPU @ 2.20GHz	256Go
lig-gpu7	2x NVIDIA Quadro RTX 6000 24Go	2x Intel Xeon E5-2623 v4 @ 2.60GHz 4C/8T	148Go
lig-gpu8	2x NVIDIA RTX A6000 48 Go GDDR6	2x Intel(R) Xeon(R) Silver 4210R CPU @ 2.40GHz	256Go
lig-gpu9	4x NVIDIA Quadro RTX 8000 48Go	2x Intel(R) Xeon(R) Silver 4210R CPU @ 2.40GHz	256Go

FIGURE 3 – Liste des serveurs "lig-gpu" du LIG et de leurs capacités

2.2 Présentation des données de DyLNet

Les données du projet DyLNet représentent 45000 heures d'enregistrements réparties sur des périodes d'une semaine par an pendant trois ans, dont 800 heures ont été annotées manuellement. Ces 800 heures d'enregistrements annotées se décomposent elles-mêmes en deux parties, avec 44 heures d'enregistrements pour les adultes (enseignants et autres personnels présents dans l'école) et environ 766 heures d'enregistrements d'enfants. Les enregistrements d'adultes sont répartis par locuteur. On dénombre un total de 14 locuteurs adultes pour les 44 enregistrements d'une heure.

Les transcriptions et annotations des enregistrements quant à elles comprennent :

- La transcription normalisée. Cette dernière est la transcription exacte de l'enregistrement sauf pour ce qui est des prénoms pouvant être prononcés dans l'audio. Ces

derniers sont anonymisés, c'est-à-dire que chaque prénom est remplacé par la mention « PRENOM ». (exemple : « dans la cour hier tous les deux euh@o *PRENOM* » .

- La transcription nettoyée, qui est le résultat de la modification de la transcription normalisée après remplacement des onomatopées, des parties incompréhensibles et des mots incomplets par des codes (dans l'ordre « ooo », « xxx » et « aaa »). Ces derniers vont permettre d'uniformiser les transcriptions. (exemple : « dans la cour hier tous les deux *ooo* PRENOM »).

- La transcription nettoyée mise sous la forme d'un étiquetage morphosyntaxique (ou POS tagging). Cette dernière pourra permettre l'étude de la construction des phrases utilisées par les enfants et leurs professeurs. (exemple : « *dans/PRP/dans la/DET :def/le cour/NOM/cour hier/ADV/hier tous/DET :pre/tout les/DET :def/le deux/NUM/@card@ ooo/NOM/ooo PRENOM/NOM/PRENOM* » .

- L'étiquetage d'activité en cours indique l'activité du porteur du microphone. Il y a cinq types d'activités possibles prévues pour l'annotation : « classe » lorsque le porteur se trouve en classe, « Récréation » lors des temps de récréations, « Hors classe » au moment des trajets dans les couloirs ou vers les toilettes, aux toilettes, à la bibliothèque, etc., « Sport » pendant le sport (gym, danse ou autre activité sportive) et « Indéterminé » quand aucune information sur l'activité en cours n'est donnée.

2.3 Speechbrain

2.3.1 Présentation et fonctionnalitées

Speechbrain est un toolkit open source basé sur Pytorch. Il a de nombreuses fonctionnalités le rendant utile pour de nombreux types de recherches en traitement automatique des langues (Ravanelli et al. [2021a]).

Speechbrain a de nombreuses fonctionnalités et peut être utilisé pour plusieurs types de tâches, selon ce que l'utilisateur veut faire. Il est possible de l'utiliser pour faire de la reconnaissance de parole comme nous cherchons à faire pour ce stage. Cependant, il est aussi possible de s'en servir pour faire de la compréhension du langage parlé, de l'identification de locuteur, de la séparation vocale ou de la segmentation de l'audio en locuteurs. Dans ces trois cas, on l'utilisera pour reconnaître les locuteurs. Il peut

aussi être utilisé pour préparer les données, pour améliorer un signal de parole, pour entraîner un modèle de langue ou pour traiter des signaux multi-microphones en combinant les signaux captés par plusieurs appareils d'enregistrement. Et, enfin, il est possible de l'utiliser pour réaliser de la traduction automatique ou de la synthèse vocale.

2.3.2 Modularité de Speechbrain

Speechbrain repose sur un ensemble de modules permettant d'accomplir différentes tâches.

Dans le domaine de la reconnaissance automatique de la parole, Speechbrain offre plusieurs modèles entraînés [2], tels que Wav2Vec2 et Whisper. Wav2Vec2 est une architecture basée sur des transformers permettant d'entraîner des modèles de façon auto-supervisée. Ce type de modèle s'avère très intéressant pour la reconnaissance automatique de la parole **??**, **?** une fois entraîné. De son côté, Whisper est une architecture basée sur un modèle de type encodeur-décodeur Transformer, et a été entraîné sur 680 000 heures de données multilingues et multitâches supervisées (OpenAI). Speechbrain permet d'intégrer facilement ces modèles entraînés pour effectuer des tâches de reconnaissance de la parole.

En ce qui concerne l'extraction de caractéristiques et l'amélioration de la parole, Speechbrain effectue automatiquement l'apprentissage de filter banks pour extraire des caractéristiques, normaliser les données et réduire le bruit et la réverbération afin d'obtenir de meilleurs résultats.

Pour des tâches telles que la reconnaissance et l'identification des locuteurs, la diarisation ou la séparation vocale, Speechbrain utilise un modèle ECAPA-TDNN (Extended Context Aggregated Parallel Attention - Time-Delay Neural Network). Ce modèle permet de capturer les relations entre les différentes trames. De plus, une implémentation Xvectors est utilisée pour différencier les locuteurs, ainsi que des librairies spécialisées.

Si le but est de faire de la synthèse de parole, Speechbrain propose différentes recettes. Par exemple, on retrouve Tacotron2 qui utilise les données de LJSpeech, ainsi que des recettes axées sur la mise au point et l'entraînement de vocodeurs, comme HiFIGAN.

2. Un modèle entraîné ou pré-appris est un modèle déjà entraîné sur de grandes quantités de données, que l'on peut réutiliser rapidement sans avoir à tout réapprendre depuis zéro

Dans le cas où l'utilisateur souhaite effectuer une identification de langue, Speechbrain utilise le modèle ECAPA-TDNN mentionné précédemment, qui a été entraîné sur VoxLingua107. Ce corpus comprend des segments de parole courts dans 107 langues différentes, extraits de vidéos YouTube. Le corpus a été spécialement conçu pour former des modèles d'identification de langues parlées.

Enfin, en ce qui concerne la traduction automatique de la parole, Speechbrain propose des recettes end-to-end utilisant des transformers. Cela permet à l'utilisateur de choisir le type d'entraînement souhaité, tel que l'attention, CTC+Attention ou la traduction de la parole avec ASR.

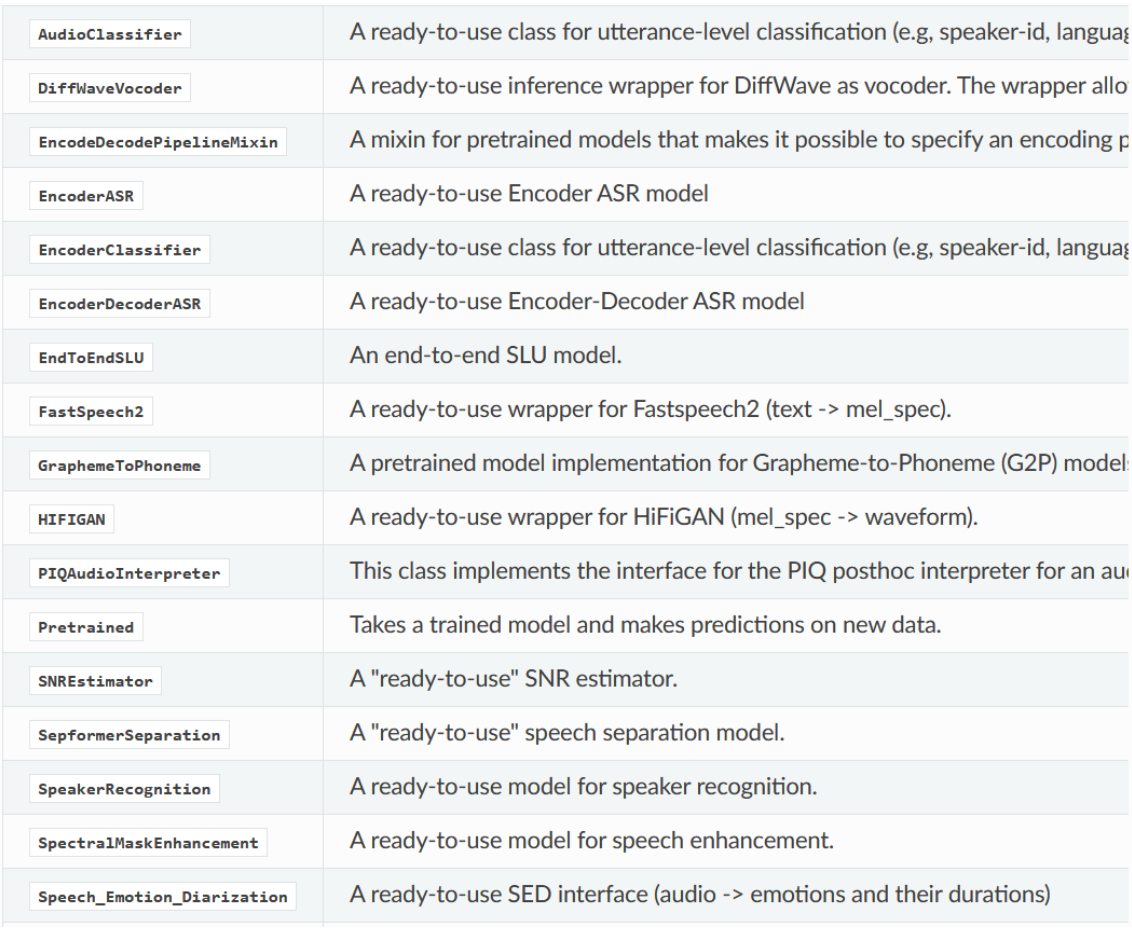
Speechbrain se distingue par son aspect modulaire, ce qui signifie que ses fonctionnalités peuvent être utilisées de manière indépendante selon les besoins de l'utilisateur. Speechbrain permet aussi l'utilisation de modules d'interface entraînés permettant plusieurs tâches (voir 4). De plus, la communauté de Speechbrain est très réactive et met à disposition de nombreuses recettes et ressources pour faciliter l'utilisation de la plateforme.

2.3.3 Performances de la reconnaissance automatique de la parole

Pour ce qui est des performances de Speechbrain sur la reconnaissance de parole, on retrouve des WER allant de 9.96%, pour de la reconnaissance de parole sur la version 6.1 du corpus CommonVoice FR (corpus de lecture de phrases en français) en utilisant Speechbrain avec le modèle wav2vec2-FR-7K-Large et une architecture wav2vec2 + CTC (Ravanelli et al. [2021b]), à 1.9%, sur libriSpeech (corpus de livres lu en anglais Panayotov and Povey [2015]) avec Speechbrain utilisant le modèle wav2vec2 (Ravanelli et al. [2021b]).

Corpus	langue	type de parole	architecture	WER
Common Voice 6.1	Français	phrases lues	wav2vec2 + CTC	9.96%
LibriSpeech	Anglais	livres lus	wav2vec2 + CTC	1.9%

TABLE 1 – Performances de la reconnaissance automatique de la parole de Speechbrain



`AudioClassifier`	A ready-to-use class for utterance-level classification (e.g, speaker-id, languag
`DiffWaveVocoder`	A ready-to-use inference wrapper for DiffWave as vocoder. The wrapper allo
`EncodeDecodePipelineMixin`	A mixin for pretrained models that makes it possible to specify an encoding p
`EncoderASR`	A ready-to-use Encoder ASR model
`EncoderClassifier`	A ready-to-use class for utterance-level classification (e.g, speaker-id, languag
`EncoderDecoderASR`	A ready-to-use Encoder-Decoder ASR model
`EndToEndSLU`	An end-to-end SLU model.
`FastSpeech2`	A ready-to-use wrapper for Fastspeech2 (text -> mel_spec).
`GraphemeToPhoneme`	A pretrained model implementation for Grapheme-to-Phoneme (G2P) model
`HIFIGAN`	A ready-to-use wrapper for HiFiGAN (mel_spec -> waveform).
`PIQAudioInterpreter`	This class implements the interface for the PIQ posthoc interpreter for an au
`Pretrained`	Takes a trained model and makes predictions on new data.
`SNREstimator`	A "ready-to-use" SNR estimator.
`SepformerSeparation`	A "ready-to-use" speech separation model.
`SpeakerRecognition`	A ready-to-use model for speaker recognition.
`SpectralMaskEnhancement`	A ready-to-use model for speech enhancement.
`Speech_Emotion_Diarization`	A ready-to-use SED interface (audio -> emotions and their durations)

FIGURE 4 – Extrait de la liste de modules entraînés proposée dans la documentation de Speechbrain

Le WER n'est pas la seule métrique pouvant être utilisée pour calculer les performances de Speechbrain, on peut citer, entre autres, des métriques telles que le CER (Character Error Rate) et le PER (Phone Error Rate) Ravanelli et al. [2021b] ; cependant, je ne parlerai dans ce rapport que de la métrique WER, car c'est celle-ci que je vais utiliser afin d'évaluer mes résultats. Les performances des systèmes de reconnaissance automatique de la parole se calculent notamment avec le WER (Word Error Rate), ou taux d'erreurs mots en français, est une mesure de l'erreur de reconnaissance de la parole ou de la transcription automatique. Ce dernier est calculé en comparant la sortie générée par le système de reconnaissance de la parole ou l'algorithme de transcription automatique avec une transcription de référence réalisée par un humain. Son calcul se fait en comptabilisant le nombre total d'erreurs, telles que les mots manquants, les mots en trop et les substitutions (un mot remplacé par un autre) puis en divisant le total d'erreurs par le nombre total de mots dans la référence via la formule suivante (Ali and Renals [2018]).

$$WER = \frac{substitutions + omissions + insertions}{nombre\_mots\_reference}$$

Les recettes speechbrain permettent d'atteindre des taux de 2 à 9%, comme le montre le tableau 1, ce qui correspond à l'état de l'art.

2.4 Expérimentations

2.4.1 Architecture du système

Pour répondre à la demande faite pour ce stage, il a été donc été choisi que j'utiliserai un modèle de Wav2vec2 appris sur le français (Baevski et al. [2021]Solène et al. [2021]). Ce dernier a démontré de bonnes performances dans le domaine de la reconnaissance automatique de la parole (Baevski et al. [2021]). J'utiliserai le modèle LeBenchmark wav2vec2-FR-7K-Large appris sur 7000 heures de paroles avec une architecture wav2vec2 + DNN + CTC. Ce type d'architecture utilise wav2vec2 qui va permettre de transformer un signal audio brut (onde sonore) en embeddings (représentations vectorielles). Il utilisera ensuite un réseau de neurones profonds (ou Deep Neural Network), ce dernier va prendre les embeddings de wav2vec2 en entrée et les transformer en prédictions de texte. Enfin, une fonction CTC (Connectionist Temporal Classification) sera utilisée. Il s'agit d'une fonction de coût utilisée afin d'entraîner le réseau de neurones profond afin qu'il puisse prédire des séquences de caractères à partir du signal audio. Cette dernière permet de gérer le décalage temporel entre le signal audio et le texte correspondant.

2.4.2 Common Voice

Dans un premier temps, afin d'apprendre à utiliser Speechbrain, mais aussi pour réaliser une première expérimentation contrôlée, j'ai commencé avec le corpus d'enregistrements du projet Common Voice FR. Lancé en juillet 2017 par l'entreprise Mozilla, avec une attention initiale sur l'anglais, le projet a été étendu à d'autres langues à partir de juin 2018. Ce dernier est une collection massivement multilingue de discours transcrits destinée à la recherche et au développement de systèmes de reconnaissance automatique de la parole Ardila et al. [2020]. Mais ce dernier peut également être utile dans d'autres domaines, tels que par exemple l'identification de langue (Ardila et al. [2020]).

J'ai utilisé la version 6.1 de Common Voice FR qui est la version française du corpus Common Voice (git). Il comprend des enregistrements de discours en français,

collectés et validés, c'est-à-dire que ces derniers ont été vérifiés et approuvés comme étant de bons exemples d'enregistrements de lecture de phrases en français. L'utilisation des outils de décodage sur ce corpus m'a permis d'explorer et de découvrir le fonctionnement de ces derniers.

Ainsi, dans le but de tester le modèle choisit, et afin de m'entraîner, j'ai réalisé un décodage des fichiers audios du corpus CommonVoice en utilisant Speechbrain et le modèle wav2vec-FR-7K-Large entraîné sur les données de CommonVoice de wav2vec2.0 PARCOLLET [2021/2022]. Le fait de tester le modèle et le décodage m'a permis de voir (et surtout de faire) les erreurs pouvant empêcher le modèle et le décodage de donner de bons résultats ou même de fonctionner, mais ce dernier m'a surtout permis de constater l'efficacité du modèle wav2vec-FR-7K-Large et de vérifier que le système tournait correctement sur mon ordinateur.

Afin de réaliser le décodage des enregistrements de CommonVoice, il a été nécessaire de récupérer les données du projet, ces dernières étant composées d'un corpus d'enregistrements audios et de leur transcription. Il a aussi fallu récupérer le système de reconnaissance automatique de parole ayant déjà été entraîné sur une partie de ces données, plus précisément sur le « Train » (voir 2.4.5 pour plus d'information sur le partitionnement). J'ai donc récupéré le modèle, ayant déjà été entraîné sur CommonVoice, sur la plateforme HuggingFace en prenant soin de prendre le bon modèle, c'est-à-dire le modèle ayant été entraîné sur Common Voice en français (Speechbrain/asr-wav2vec2-commonvoice-fr PARCOLLET [2021/2022]). Ce dernier est censé permettre un taux d'erreur mots (WER), lors du décodage, de 9,96 %, ainsi, un résultat trop différent de celui prévu signifierait qu'une erreur a été faite.

Une fois le système d'ASR récupéré, il était nécessaire de récupérer aussi les données de Common Voice Fr et c'est à ce point qu'apparait ma première erreur. En effet, après avoir téléchargé les enregistrements audios et les transcriptions de ces derniers et une fois le décodage effectué, il s'est avéré que la version du corpus Common Voice Fr que j'avais n'était pas celle ayant été utilisée pour le décodage dont les résultats étaient donnés dans la littérature, une simple vérification du fichier « README.md » du modèle aurait pu m'éviter l'erreur suivante : j'avais téléchargé Common Voice Corpus 7.0 alors qu'il était précisé dans ledit fichier que la version qu'il fallait était la version 6.1. Ainsi, le modèle ne pouvait pas me donner les résultats escomptés. Une fois le changement de version de Common Voice 7 pour la version 6.1 effectué, une erreur subsistait. Le résultat obtenu était un WER de 6,8 %. Il est vrai que ce dernier était meilleur que celui présenté sur la plateforme hugging Face, mais, du fait de sa différence de plus de 3 % avec le résultat prévu, il signifiait qu'une erreur était toujours présente. Le fait que le taux d'erreurs mots soit inférieur à celui prévu

donnait une bonne indication de la localisation de cette dernière, puisque, n'ayant pas touché au modèle de reconnaissance automatique de parole, la différence entre le résultat obtenu et celui escompté ne pouvait venir que d'une erreur dans les fichiers traités. Et c'était le cas, à la suite d'une erreur de manipulation, le décodage ne s'effectuait pas sur la partie « test » des audios, mais sur la partie « train » (voir 2.4.5), c'est-à-dire la partie sur laquelle le modèle avait été entraîné (le résultat obtenu ne pouvait alors qu'être meilleur que celui prévu).

Ainsi, après avoir corrigé les différentes erreurs, le taux d'erreur mots obtenu était 9,88%. Ce dernier reste inférieur à celui annoncé par Speechbrain, mais la différence est négligeable et permet d'affirmer le bon fonctionnement du modèle et la bonne manipulation des fichiers. De plus, l'obtention de ce résultat m'a permis de vérifier ma capacité à faire fonctionner ce système. Ainsi, j'allais pouvoir utiliser le même type de modèle que pour ce dernier lors des expérimentations sur les données du corpus du projet DyLNet, et ce, à la fois en tant que modèle de référence et en tant que modèle à fine-tuner [3].

J'ai aussi comparé les performances obtenues sur le corpus de Common Voice Fr avec Speechbrain avec celles de Whisper, un système développé par Open AI et rendu public en 2022. Comme dit précédemment, ce dernier est entraîné sur des quantités massives (680000 heures) de données prises sur internet. J'ai donc comparé les résultats des deux décodages effectués avec Whisper et Speechbrain sur les parties test et train de Common Voice. Pour la partie train Speechbrain utilisant une architecture wav2vec2+CTC+attention obtient un WER de 7.4% contre 19.2% pour whisper et pour la partie test Speechbrain obtient un WER de 9,89% alors qu'en utilisant whisper, on obtient un WER de 23.5%. Cette comparaison confirme le choix de l'utilisation de Speechbrain par rapport à whisper.

2.4.3 DyLNet

Après avoir établi la validité opérationnelle du modèle entraîné sur les données du corpus Common Voice Fr (et ce même si cette dernière était déjà certaine, car évaluée par LeBenchmark Solène et al. [2021]), il était temps d'aborder l'étape cruciale de ce stage : le lancement de l'entraînement d'un système de reconnaissance vocale de l'enseignant, en appliquant les connaissances acquises lors des tests sur le projet Common Voice, aux données du projet DyLNet.

3. Le fine-tuning consiste à continuer à entraîner un modèle pré-appris sur de nouvelles données spécifiques, afin de l'adapter à une tâche précise

Cependant, avant de choisir et de lancer les expérimentations, il convient de récupérer les données du projet DyLNet, de les partitionner et de les préparer, pour enfin mettre en place les différentes expérimentations allant être lancées.

2.4.4 Récupération des données

La récupération des données du projet DyLNet s'est faite sur le serveur lig-theradia-data, serveur sur lequel sont actuellement stockées toutes les données du projet DyLNet. Ces données se composent de 800 heures d'enregistrements et de leurs transcriptions. La difficulté de la récupération des données résidait, d'une part, dans la très grande quantité de dossiers, répartie sous forme arborescente. En effet, les données du projet DyLNet sont réparties sur le serveur lig-theradia-data en trois dossiers, un par année d'enregistrements. Chaque dossier contient ensuite quatre ou cinq sous-dossiers (en fonction de l'année), qui à leur tour contiennent plusieurs dizaines de dossiers, un par locuteur. Chaque dossier de locuteur contient entre un et six sous-dossiers contenant les fichiers audio, un fichier Textgrid et plusieurs fichiers EAF. D'une autre part, le sujet du stage portant sur la parole de l'enseignant, il a été nécessaire de trouver, parmi l'ensemble des locuteurs, ceux étant des adultes. Pour faire cela, le moyen le plus simple a été d'écouter un extrait d'enregistrement de chaque locuteur afin de repérer les quatorze locuteurs adultes du corpus.

Ainsi, une fois les identifiants des locuteurs adultes repérés, j'ai pu effectuer la récupération des données audio et de transcription de ces derniers afin de les transférer sur mon serveur de travail et pouvoir faire le partitionnement de ces données.

2.4.5 Partitionnement

Un partitionnement des données est nécessaire lorsque l'on souhaite entraîner ou fine-tuner [4] un modèle de reconnaissance automatique de parole. Ce dernier a pour but d'obtenir trois parties contenant chacune des données différentes. Traditionnellement, ces trois parties sont nommées « Train », « Dev » et « Test » et ont chacune leur utilité dans le cadre de l'entraînement ou du fine-tuning.

4. pour rappel, il s'agit du fait d'ajuster les paramètre d'un modèle pré-entraîner sur de nouvelles données

- La partie Train, pour entraînement, a pour but de servir à entraîner ou fine-tuner le modèle de reconnaissance de la parole. Elle permettra d'ajuster les paramètres de ce dernier en utilisant un ensemble de données. Dans notre cas, il s'agira d'un ensemble d'enregistrements et leurs transcriptions venant de plusieurs locuteurs. Le modèle est exposé à ces données pour apprendre à reconnaître les motifs et les caractéristiques des discours.

- La partie Dev, pour développement ou validation, sert à ajuster les hyperparamètres du modèle. Ces derniers sont les paramètres qui contrôlent le processus d'apprentissage du modèle, tels que la taille du lot, le taux d'apprentissage, etc. Cette partie permet de sélectionner les meilleurs hyperparamètres pour améliorer les performances du modèle.

- La partie Test est utilisée pour évaluer les performances du modèle de reconnaissance de la parole sur des données n'ayant pas été vues auparavant (donc des données ne se trouvant ni dans la partie Train ni dans la partie Dev). Ces données de test sont utilisées dans le but d'estimer la capacité qu'a le modèle de reconnaissance automatique de parole à généraliser et à reconnaître correctement les discours dans des conditions réelles.

Ainsi, une fois l'ensemble des données d'enregistrements et de transcriptions des enseignants récupérées, il a fallu décider comment faire le partitionnement. En effet, les données d'enregistrement d'enseignants et personnel adulte de l'école comptabilisent 44 heures pour quatorze locuteurs. Parmi ces locuteurs, il est important de préciser que seulement trois sont des locuteurs masculins. Cette distinction est importante car le modèle a été entraîné avec une proportion plus élevée de voix féminines que de voix masculines. Il est possible que cela puisse affecter les résultats du modèle lorsqu'il s'agit de reconnaître les voix masculines. Pour garantir le meilleur entraînement et fine-tuning possible, il a été décidé de partitionner les données d'enregistrement. La partie Train contient donc 80% du temps d'enregistrement, tandis que la partie Dev et la partie Test contiennent chacune 10% du temps d'enregistrement. De plus, chaque partie est composée de locuteurs qui n'apparaissent pas dans les autres parties. Il a été spécifié que la partie Test doit inclure au moins un locuteur masculin pour évaluer les performances du modèle sur les voix masculines. Les deux autres locuteurs masculins ont été placés dans la partie Train afin de garantir que le modèle soit également entraîné sur des voix masculines. Une fois le temps de parole complet de chaque locuteur comptabilisé, j'ai sélectionné pour chacun d'eux la partie adéquate afin de respecter au maximum les proportions choisies. Ainsi, j'ai pu me rapprocher de ces proportions en obtenant environ 82% du temps de parole dans la partie Train, environ 9% dans la partie Dev, et environ 9% dans la partie

Test. Cela équivaut à trente-six heures d'enregistrement pour la partie Train, et quatre heures pour les parties Dev et Test. J'ai également veillé à respecter le fait que chaque locuteur n'apparaisse que dans une seule partie. De plus, j'ai respecté la répartition des voix masculines en plaçant deux locuteurs masculins dans la partie Train et un dans la partie Test.

2.4.6 Récupération des transcriptions et création des fichiers JSON

Lorsqu'on observe les transcriptions présentes dans les données du projet DyLNet, on remarque qu'elles sont toutes sous formats « eaf », c'est-à-dire qu'elles sont sous le format du logiciel ELAN, le logiciel ayant été utilisé afin de réaliser la transcription de chaque audio. Cependant, les programmes utilisés pour l'entraînement et le fine-tuning ne sont pas prévus pour aller récupérer les transcriptions des audios sur un fichier « eaf » mais sur un fichier « JSON » respectant une forme précise.

"36-20200213-080000-1013-41E0-03-part02-R_VB_2960400_2962800" :
{
"spk_id" : "36-20200213-080000-1013-41E0-03-part02-R_VB",
"start_seg" : 2960.4,
"end_seg" : 2962.8,
"duration" : 2.4,
"wav" : "partitions/Train_files/audios/36-20200213-080000-1013-41E0-03-part02-R.wav",
"activité" : "En classe",
"wrd" : "Y A LÀ DES PUZZLES APRÈS",
"char" : "Y _ A _ L À _ D E S _ P U Z Z L E S _ A P R È S"
},
"36-20200213-080000-1013-41E0-03-part02-R_VB_2963060_2965340" :
{
"spk_id" : "36-20200213-080000-1013-41E0-03-part02-R_VB",
"start_seg" : 2963.06,
"end_seg" : 2965.34,
"duration" : 2.28,
"wav" : "partitions/Train_files/audios/36-20200213-080000-1013-41E0-03-part02-R.wav",
"activité" : "En classe",
"wrd" : "OOO JE SAIS PAS SI C EST CEUX LÀ MAIS OOO",
"char" : "O O O _ J E _ S A I S _ P A S _ S I _ C _ E S T _ C E U X _ L À _ M A I S _ O O O"
},
"36-20200213-080000-1013-41E0-03-part02-R_VB_2972980_2974060" :
{
"spk_id" : "36-20200213-080000-1013-41E0-03-part02-R_VB",
"start_seg" : 2972.98,
"end_seg" : 2974.06,
"duration" : 1.08,
"wav" : "partitions/Train_files/audios/36-20200213-080000-1013-41E0-03-part02-R.wav",
"activité" : "En classe",
"wrd" : "OOO SINON APRÈS OUAIS",
"char" : "O O O _ S I N O N _ A P R È S _ O U A I S"
},
"36-20200213-080000-1013-41E0-03-part02-R_VB_3005600_3007200" :
{
"spk_id" : "36-20200213-080000-1013-41E0-03-part02-R_VB",
"start_seg" : 3005.6,
"end_seg" : 3007.2,
"duration" : 1.6,
"wav" : "partitions/Train_files/audios/36-20200213-080000-1013-41E0-03-part02-R.wav",
"activité" : "En classe",
"wrd" : "OOO ÇA OOO",
"char" : "O O O _ Ç A _ O O O"
},

FIGURE 5 – Extrait du fichier Train.JSON

L'extrait du fichier JSON présenté page précédente (2.4.6) contient toutes les informations nécessaires à l'entraînement / fine-tuning, dont : l'identifiant du locuteur (spk_id), les temps de début et de fin de l'extrait (start_seg et end_seg), la durée de l'extrait (duration), le chemin d'accès au fichier audio (wav), l'activité en cours (activité), la transcription (wrd) et enfin la transcription séparée en caractères (char).

Ainsi, pour obtenir un fichier JSON semblable et donc récupérer toutes les informations nécessaires à la création des fichiers JSON et notamment la transcription de chaque moment de parole, il a fallu utiliser trois programmes. Dans un premier temps, afin de récupérer rapidement les informations, j'ai utilisé un programme (code) permettant de récupérer toutes les informations, qui seront utiles, du fichier Elan et de les ajouter dans un fichier texte compilant les informations de tout les enregistrement de la partition. Ce programme récupère donc le temps de début et le temps de fin du segment de parole (en millisecondes), l'activité en cours et la transcription normalisée.

```
3446152 3448128 En classe       ça c'est fait. ooo ooo voilà c'est bien ça.
3448773 3450773 En classe       tu mets à sécher sur la petite table là.
3451328 3453820 En classe       PRENOM tu vas dehors? PRENOM tu t'habilles.
3454061 3456075 En classe       dans l'autre sens les oreilles s'il te plaît.
3458758 3461428 En classe       tu as fini? alors va t'habiller tu peux sortir.
3465285 3468044 En classe       ça PRENOM c'est bon. fais sécher.
3471997 3473460 En classe       oui PRENOM tu as besoin d'aide?
3474019 3474389 En classe       oui?
3476359 3477625 En classe       il me dit toujours ouais PRENOM.
3477965 3479108 En classe       ouais ouais ouais ouais.
3480125 3481091 En classe       ouais ouais ouais.
3482261 3484837 En classe       ooo PRENOM ça fait trop tout ça oooooo.
3485600 3486748 En classe       on va mettre que le gilet.
3486950 3487251 En classe       tiens.
3488532 3491492 En classe       et tu vas aller remettre ta veste au portemanteau.
3492113 3494434 En classe       parce que ça fait trop là. oui c'est très bien ooo.
3495601 3498005 En classe       PRENOM. on fait sécher là c'est très bien. voilà.
3499630 3500354 En classe       allez PRENOM.
3501374 3503029 En classe       vous allez vous habiller vous sortez.
3504357 3504834 En classe       vas y.
3509295 3511133 En classe       je vais appeler la maman de PRENOM quand même.
```

FIGURE 6 – Extrait d'un fichier texte sur lequel ont été récupérées les informations utiles du fichier Elan (temps de début et de fin des extraits, activité et transcription)

Une fois cette étape réalisée, il est maintenant nécessaire de récupérer les chemins d'accès aux différents fichiers audio. Pour rappel, le chemin d'accès d'un fichier est l'emplacement spécifique du fichier dans le système de fichiers, indiquant les répertoires et sous-répertoires nécessaires pour accéder à ce dernier. La récupération de ces chemins d'accès se fait via l'utilisation d'un autre programme (code) qui va par-

courir tous les dossiers et sous dossiers à partir d'un endroit donner et va inscrire, dans un fichier texte, le chemin de chaque fichier de transcription qu'il trouve (ici des fichiers en « .txt »). Ces derniers, mis à part pour l'extension qui est « .txt » au lieu de « .wav », ont le même nom que les fichiers d'enregistrement. Ainsi lorsqu'ils seront utilisés pour la mise au point du fichier JSON final, le programme pourra utiliser les chemins de ce fichier pour récupérer à la fois la transcription et l'enregistrement (en changeant l'extension pour « .wav »).

```
partitions/Train_files/transcriptions/21-20181009-070000-1033-4152-03-part02-R_VB.txt
partitions/Train_files/transcriptions/21-20181010-070000-1033-4152-04-part02-R_VB.txt
partitions/Train_files/transcriptions/21-20190513-070000-1033-4152-00-part02-R_VB.txt
partitions/Train_files/transcriptions/21-20181011-070000-1033-4152-05-part02-R_VB.txt
partitions/Train_files/transcriptions/21-20190515-070000-1033-4152-04-part02-R_VB.txt
partitions/Train_files/transcriptions/21-20190516-070000-1033-4152-06-part02-R_VB.txt
partitions/Train_files/transcriptions/21-20190517-070000-1033-4152-08-part02-R_VB.txt
partitions/Train_files/transcriptions/21-20181011-063316-1073-41D0-00-part01-R_VB.txt
partitions/Train_files/transcriptions/21-20181011-070000-1073-41D0-00-part02-R_VB.txt
partitions/Train_files/transcriptions/21-20181011-080000-1073-41D0-00-part03-R_VB.txt
partitions/Train_files/transcriptions/21-20190617-070000-1073-41C0-04-part02-R_VB.txt
partitions/Train_files/transcriptions/21-20190618-070036-1073-41C0-06-part01-R_VB.txt
partitions/Train_files/transcriptions/21-20190619-070214-1073-41C0-08-part01-R_VB.txt
partitions/Train_files/transcriptions/21-20190620-080000-1073-41C0-09-part02-R_VB.txt
partitions/Train_files/transcriptions/22-20181008-080000-1041-4144-00-part03-R_VB.txt
partitions/Train_files/transcriptions/22-20190513-070000-1041-4144-00-part02-R_VB.txt
partitions/Train_files/transcriptions/22-20181010-080000-1041-4144-03-part02-R_VB.txt
partitions/Train_files/transcriptions/22-20190515-070000-1041-4144-02-part02-R_VB.txt
partitions/Train_files/transcriptions/22-20181012-080000-1041-4144-05-part02-R_VB.txt
partitions/Train_files/transcriptions/22-20190516-070000-1041-4144-03-part02-R_VB.txt
partitions/Train_files/transcriptions/22-20181008-080000-1260-4133-00-part03-R_VB.txt
partitions/Train_files/transcriptions/22-20190513-080000-1260-4133-00-part03-R_VB.txt
partitions/Train_files/transcriptions/22-20181010-080000-1260-4133-02-part02-R_VB.txt
partitions/Train_files/transcriptions/22-20190515-080000-1260-4133-04-part03-R_VB.txt
partitions/Train_files/transcriptions/22-20181011-080000-1260-4133-03-part03-R_VB.txt
partitions/Train_files/transcriptions/22-20190516-080000-1260-4133-05-part03-R_VB.txt
```

FIGURE 7 – Extrait d'un fichier texte sur lequel ont été récupérées les chemins d'accès aux données de transcriptions d'enregistrements

Enfin, une fois toutes les données nécessaires à la création du fichier JSON réunies, il est alors possible de lancer un dernier programme (script). Celui-ci va récupérer les informations du fichier texte contenant le temps de début et de fin du segment de parole, l'activité en cours et la transcription, ainsi que le chemin du fichier audio afin de les ajouter dans un fichier JSON. Ce dernier contiendra alors huit informations par segment de parole : l'identifiant du locuteur, le timecode de début du segment et celui de fin du segment de parole, la durée du segment (qui est calculée par le programme), le chemin d'accès au fichier audio contenant le segment de parole, l'activité en cours, la transcription normalisée et la transcription découpée en caractères.
Ainsi, grâce à ce fichier JSON, le programme permettant l'entraînement ou le fine-tuning aura accès à la transcription du segment de parole et pourra retrouver le

fichier audio et accéder au segment correspondant à cette transcription.

2.4.7 Expérimentations passées

En 2022, Hà Nguyen a déjà réalisé un travail concernant l'application et l'adaptation de modèles de reconnaissance automatique de la parole sur des données audio annotées du projet DyLNet. Pour ce travail, Hà Nguyen a appliqué le modèle ASR wav2vec2 entraîné sur les données de Common Voice (asr-wav2vec2-commonvoice) aux enregistrements audio du projet DyLNet, mais le taux d'erreur initial était de 50%. Pour remédier à cela, il a procédé à l'adaptation du modèle en le fine-tunant sur les données de DyLNet partitionnés. Après 10 epochs d'entraînement, le taux d'erreur de mots est tombé à 22% sur la partition de test. Cette adaptation démontre l'efficacité de personnaliser les modèles ASR génériques pour des données spécifiques. Cette avancée ouvrent des perspectives prometteuses pour les résultats des modèles mis en place durant mon stage. Afin d'essayer d'obtenir de meilleurs résultats, j'ai fait le choix de ne pas reprendre le partitionnement réalisé par Hà (dont le partitionnement était de 38% pour le Train, 34% pour le Test et 28% pour le Dev) afin d'en avoir un plus équilibré.

2.4.8 Expérimentations sur DyLNet

À partir du moment où toutes les données nécessaires ont été ajoutées dans les fichiers JSON correspondant (Train.Json pour les données d'entraînement, Dev.Json pour les données de validation et Test.Json pour les données de test) il est alors possible de lancer un décodage, un entraînement (avec ou sans fine-tuning d'un modèle pré-appris) à partir de ces dernières. Cependant, avant de lancer tout entraînement ou fine-tuning, il faut choisir le ou les modèles à utiliser, ainsi que les paramètres pour ces derniers. Comme vu précédemment dans la partie sur Common Voice (2.4.2) au vu des performances du système de reconnaissance automatique de la parole issu d'HuggingFace entraîné sur les données du projet Common Voice FR, nous avons fait le choix d'utiliser ce dernier pour les expérimentations 0, 3 et 4 du tableau (2), mais pas uniquement celui-ci, car, pour mettre le plus de chance de notre côté en vue de l'obtention des meilleurs résultats possibles, il était nécessaire d'entraîner de nouveaux systèmes directement avec les données du projet DyLNet et c'est ce que j'ai fait pour les expérimentations 1 et 2 du tableau (2).

Numéro de l'expérimentation	Nom	Freeze [5]	Description
0	Wav2vec2-CV	OUI	Wav2vec2-FR-7K-Large entraîné sur les données de Common Voice (modèle déja entraîné)
1	wav2vec2+DyLNet	OUI	Wav2vec2-FR-7K-Large que j'ai entraîné sur les données de DyLNet
2	wav2vec2-CV+DyLNet	OUI	Wav2vec2-FR-7K-Large entraîné sur les données de Common Voice que j'ai fine-tuné sur les données de DyLNet
3	wav2vec2+DyLNet+défreeze	NON	Wav2vec2-FR-7K-Large que j'ai entraîné sur les données de DyLNet
4	wav2vec-CV+DyLNet+défreeze	NON	Wav2vec2-FR-7K-Large entraîné sur les données de Common Voice que j'ai fine-tuné sur les données de DyLNet en mode « défreeze »

TABLE 2 – Tableau des expérimentations

Tout d'abord, le premier choix ayant été fait a été de réutiliser le système de reconnaissance automatique de la parole issu d'HuggingFace entraîné sur les données du projet Common Voice FR sans fine-tuning (exactement comme dans l'expérimentation sur Common Voice 2.4.2) afin d'observer les résultats de ce dernier sur d'autres données que celles initialement prévues.

Ensuite, le second modèle choisit est le modèle wav2vec2-FR-7K-Large, il s'agit du même modèle que celui utilisé pour le modèle entraîné sur les données de Common Voice, c'est-à-dire le modèle wav2vec2 dans sa version large (entraînée sur 7600 heures de paroles en français), cependant ce dernier, à l'inverse du modèle utilisé pour Common Voice, n'est pas un modèle ayant subi un entraînement (autre que l'entraînement initial du modèle), ainsi, contrairement au système utilisé pour commonvoice, nous n'utilisons pas de système d'ASR tout prêt, mais en entraînons un grâce aux données dylnet et au modèle 7k pré-appris par l'équipe du projet LeBenchmark.

Pour le troisième modèle, nous avons choisi de réutiliser le système de reconnaissance automatique de parole utilisé lors du décodage des données de Common Voice, le système d'ASR issu d'HuggingFace ayant été entraîné sur ces dernières. Cependant, ce dernier sera fine-tuné sur les données de DyLNet afin d'obtenir de meilleurs résultats.

Le quatrième modèle utilisé sera le même que le second, c'est-à-dire avec le modèle wav2vec2-FR-7K-Large et un entraînement sur les données du corpus DyLNet, mais pour celui-ci le paramètre « freeze » de wav2vec2 ne sera pas activé, ainsi, les couches du modèle ne seront pas figées pendant l'entraînement, ce qui peut potentiellement permettre au modèle d'apprendre des caractéristiques plus adaptées aux données d'entrée et alors d'améliorer les performances du système de reconnaissance automatique de parole. Cependant, le déblocage de ces couches entraîne une demande de ressource (puissance de calcul) bien plus importante.

Enfin, pour le cinquième et dernier système, il s'agit une nouvelle fois du modèle Wav2vec2-FR-7K-Large ayant été entraîné sur les données du projet Common Voice et fine-tuné sur les données du projet DyLNet, cependant, de la même façon que le précédent modèle, l'hyperparamètre « freeze » est désactivé de manière que les couches du modèle ne seront pas figées pendant l'entraînement.

Ainsi, avec ces cinq expérimentations, nous espérons obtenir un modèle ayant les meilleurs résultats possibles dans la transcription automatique de parole sur les données de paroles d'adulte du projet DyLNet.

2.4.9 Entraînement des modèles

Une fois les différentes expérimentations choisies, il est temps d'entamer la phase la plus étendue de ce stage. Cette étape est caractérisée à la fois par la diversité des tâches à accomplir et par leur durée de réalisation, en faisant ainsi la partie qui a demandé le plus de temps en raison de son importance significative. En effet, parmi les cinq expérimentations évoquées précédemment (voir section 2.4.8), quatre d'entre elles nécessitaient soit un fine-tuning, soit un entraînement. Pour ce faire, diverses étapes ont été entreprises, que ce soit pour le fine-tuning ou pour l'entraînement de chaque modèle. Ces étapes impliquaient la création d'un script Python pour extraire les données d'entraînement ou de fine-tuning, la mise en place d'un fichier YAML contenant les références aux fichiers nécessaires ainsi que les hyperparamètres pour l'entraînement/fine-tuning, et la création d'un script bash pour lancer les scripts Python en conjonction avec les fichiers YAML correspondants. Une fois chaque ensemble de fichiers prêt, les processus d'entraînement ou de fine-tuning ont pu être initiés sur les serveurs du LIG. Pour ce faire, la commande décrite dans la section "Linux / Serveurs" de la troisième partie (voir section 2.1) a été employée, permettant ainsi de solliciter l'ordonnanceur du LIG pour l'utilisation de deux unités de traitement graphique (GPU).

Après plusieurs essais et après avoir écouté les conseils de mes collègues me disant qu'au vu de ma faible quantité de données, je pouvais avoir un nombre d'epochs élevé, j'ai décidé que mes entraînements et fine-tuning aurait un nombre d'époques compris entre 50 et 100 en fonction du modèle [6], favorisant un entraînement optimal, conduisant ainsi à des performances accrues du modèle. Cependant, plus celui-ci est élevé, plus la durée de l'entraînement sera long, ainsi, pour un nombre d'époques compris entre soixante-dix et cent époques, la durée des entraînements pouvait dépasser les 90 heures. Ainsi, pour une fourchette d'époques allant de soixante-dix à cent, j'ai opté pour une réservation de deux GPU sur une période de quatre-vingt-dix heures (cent heures étant le temps de réservation maximal autorisé, les demandes de réservation mettais plus de temps à démarrer). Une fois l'entraînement ou le fine-tuning lancé, la seule chose possible à faire était d'attendre que celui-ci se termine, ou que la réservation de GPU arrive à son terme, car lors de la demande d'utilisation d'un ou plusieurs GPU, il est nécessaire d'indiquer la durée maximale d'utilisation de ces derniers, une fois la durée de réservation arrivée à échéance le script est stoppé, et ce, peu importe qu'il soit terminé ou non. Cependant, le script Python utilisé pour initier l'entraînement ou le fine-tuning du modèle offre la possibilité de sauvegarder l'apprentissage toutes les 30 minutes. C'est ce que l'on appelle un checkpoint. Par conséquent, même si le script est interrompu, que ce soit en raison de l'expiration de la réservation GPU ou pour toute autre raison, les progrès réalisés jusqu'à la dernière époque complète seront préservés et accessibles.

Comme énoncé précédemment, chaque entraînement et fine-tuning a nécessité plusieurs heures de travail, temps principalement passé à essayer de faire fonctionner les scripts et à l'exploration des hyperparamètres en vue d'obtenir les meilleurs résultats. Cependant, j'ai consacré un temps considérablement plus important à un modèle spécifique : le modèle wav2vec2+dylnet. Ce modèle a été marqué par un problème persistant tout au long de cette partie finale du stage, occasionnant des retards. Surtout que le non-fonctionnement de ce dernier impliquait par la même occasion le non-fonctionnement du modèle défreezé (cf. 2.4.8) puisque ce dernier, fonctionnant sur les mêmes données que le modèle précédemment cité, se retrouvait confronté au même problème. Ces deux modèles comportait une erreur que je n'ai pas réussi à corriger, ni même à localiser. L'erreur était la suivante : au lancement du script d'entraînement, ce dernier, m'informe, via un message d'erreur, que je n'ai pas assez de mémoire disponible sur les GPU, et cela, même lorsque j'ai réduit au maximum la quantité de données à traité, le taux d'apprentissage ou learning rate [7]

6. une époque ou epoch est une itération complète de l'ensemble de données d'entraînement lors de l'apprentissage d'un modèle

7. il s'agit d'un hyperparamètre qui contrôle la taille des pas pris lors de la mise à jour des paramètres d'un modèle pendant son entraînement

et la taille du batch ou lot [8]. L'erreur n'étant visiblement pas liée à la quantité de données, mais à un autre point que je ne trouvais pas, et ce, malgré une grande quantité d'essais et de recherches infructueuses.

2.4.10 Hypothèses

En me basant sur mon raisonnement personnel ainsi que sur le travail de Hà Nguyen, il est possible de faire quelques hypothèses sur les modèles allant donner de mauvais résultats, ceux allant donner les meilleurs résultats et pourquoi.
Si on prend en compte le fait que la quantité de données de parole d'adulte ayant été transcrite reste très faible, puisque qu'elle ne comporte que quarante-quatre heures d'audio, il est alors possible d'émettre l'hypothèse, pour les deux modèles entraînés uniquement sur les données de DyLNet, de résultats peu convaincants. En effet, du fait de la faible quantité de données d'entraînement, le modèle Wav2vec2-FR-7K-Large entraîné sur les données de DyLNet et sa version défreezée n'aurons pas assez de données d'entraînement pour que le modèle soit suffisamment bien entraîné.
Si on observe le travail réalisé par Hà, et tout particulièrement les résultats de ce dernier, il est alors raisonnable de supposer que les résultats des modèles fonctionnant sur un modèle ayant été entraînée sur les données de Common Voice et fine-tuné sur les données de DyLNet (modèle freeze et modèle défreezé) seront peu éloignés de ceux obtenus par Hà, ainsi, son modèle final, pour lequel il obtient un taux d'erreurs mots de 22% après seulement dix epochs et qui n'est autre que le modèle entraîné sur Common Voice, fine-tuné sur les données Train de DyLNet et defreezé que j'utilise aussi dans ma cinquième expérimentation pourrait être le modèle obtenant les meilleurs résultats parmi tous les autres.

8. lors de l'entraînement d'un modèle, au lieu de présenter un seul exemple à la fois, on regroupe un certain nombre d'exemples en un seul lot, puis on effectue les calculs nécessaires pour ajuster les poids du modèle en fonction de ces exemples regroupés, la taille du batch est donc le nombre d'exemples dans le lot

3 Résultats finaux et discussion

3.1 Résultats

Une fois les entraînements terminés et les résultats de chaque modèle obtenus, il a été possible de comparer ces derniers. Dans un premier temps entre eux, puis de les comparer à la littérature et aux résultats obtenus par Hà NGUYEN qui, pour rappel, a aussi participé à la mise en place d'un système automatique de la parole de l'enseignant en école maternelle pour les besoins du projet DyLNet.

Pour commencer, la première expérimentation réalisée, utilisant le modèle wav2vec2-FR-7K-large pré-entraîné sur les données du projet Common Voice Fr, a obtenu un taux d'erreurs mots de 29.98%. Ce résultat n'est pas un résultat que l'on pourrait qualifier de bon. Cependant, en prenant en compte le fait qu'il s'agisse d'un modèle pré-entraîné sur un autre type de données que celle pour lequel il a ici été utilisé, ce résultat n'est pas mauvais non plus et laisse alors présager de bons résultats lors des expérimentations suivantes auquel l'entraînement sera fait sur les données de DyLNet. En plus d'un simple décodage, j'ai aussi effectué un calcul du taux d'erreurs mots en fonction du locuteur et en fonction de l'activité. Ainsi, pour la partie de test qui contient les enregistrements de deux locuteurs, le WER d'un locuteur est de 20.11% contre 31.55% pour le second. Celui ayant le meilleur résultat est le locuteur masculin, cependant, il prononce beaucoup moins de mots que l'autre locuteur (7454 contre 46 635), cette différence de quantité de parole fait qu'il n'est pas possible d'avancer que l'ASR réalise de meilleures performances sur la parole de locuteur masculin. Pour ce qui est du WER selon l'activité, les résultats obtenus sont tout aussi distincts.

WER	Activité
28.88%	En classe
33.65%	Hors classe
25.40%	Indéterminé
20.94%	Récréation
31.80%	Sport

Table 3 – Tableau des résultats selon l'activité

Ces résultats permettent de se rendre compte de la différence des qualités d'enregistrements ou du moins de déduire les activités durant lesquelles il y a le plus

de bruit, ici, c'est visiblement hors classe, c'est-à-dire durant les trajets dans les couloirs qu'il y a le plus de bruit.

Pour l'expérimentation sur le modèle wav2vec2-FR-7K-large, pré-entraîné sur les données du projet Common Voice Fr et fine-tuné sur la parole d'enseignants du projet DyLNet le résultat a été plus surprenant. Je m'attendais à obtenir de meilleurs résultats que lors du décodage utilisant le modèle uniquement pré-entraîné sur les données de Common Voice Fr du fait du fine-tuning. Cependant, le taux d'erreurs mots de cette expérimentation est de 35.63%, soit plus de 5% supérieur à celui obtenu avec la première expérimentation. Ce résultat peut nous faire arriver à trois conclusions possibles, soit je me suis trompé quelque part, soit le fait d'uniquement fine-tuner le modèle pré-entraîné n'est pas suffisant et lui ajoute des erreurs au lieu de les réduire, ou alors les hyperparamètres n'étais pas les bons.
Pour ce qui est de la troisième raison possible menant à l'obtention de ce résultat, je ne pense pas qu'elle soit réaliste. En effet, pour celui-ci, les hyperparamètres (les hyperparamètres sont les paramètres qu'on définit avant l'entraînement d'un modèle, comme le taux d'apprentissage, le nombre d'epochs ou la taille des batches, dans le but d'optimiser les performances du modèle pour la tâche de fine-tuning ou d'entraînement visée) n'ont pas de valeur extravagante, le nombre d'epochs était de 50 et la taille du batch de 4, ce qui me permet de douter du fait que ce résultat décevant soit lié aux hyperparamètres.

C'est avec l'expérimentation sur le modèle wav2vec2-FR-7K-large, pré-entraîné sur les données du projet Common Voice Fr et fine-tuné sur la parole d'enseignants du projet DyLNet mais cette fois-ci en changeant la valeur de l'hyperparamètre responsable du freeze de wav2vec2 de «True» vers «False», que le taux d'erreurs mots a été le plus bas. J'ai d'abord commencé par lancer cette expérimentation avec une taille de batch de 5 et pour 60 epochs. Cependant, en observant les variations de résultats epochs par epochs, j'ai remarqué que le modèle était toujours en train de s'améliorer même après 60 epochs. À 60 epochs cette expérimentation, obtenait déjà un WER de 20%, j'ai alors pris la décision de relancer cette dernière jusqu'à 100 epochs, ce qui peut paraitre beaucoup, mais a été utile. C'est à partir de l'epochs 85 que le fine-tunage du modèle a cessé d'améliorer le résultat de ce dernier. Ainsi, pour cette expérimentation, après 85 epochs, j'ai obtenu un taux d'erreurs mots de 17.89%, ce dernier est le meilleur parmi toutes les expérimentations ayant fonctionné.

Nom	Freeze	Résultat (WER)
Wav2vec2-CV	OUI	29.98%
wav2vec2+dylnet	OUI	
wav2vec2-CV+dylnet	OUI	35.63%
wav2vec2+dylnet+défreeze	NON	
wav2vec-CV+dylnet+défreeze	NON	17.89%

TABLE 4 – Tableau des résultats

Comme vu précédemment (Cf. 2), il était aussi prévu deux autres expérimentations utilisant le modèle wav2vec2-FR-7K-large. Ces deux expérimentations devait être directement entraînées sur les données de DyLNet, l'une avec l'hyperparamètre «freeze» activé et l'autre non. Cependant, il m'a été impossible de lancer ces deux entraînements (Cf. 2.4.9) et ce malgré le temps passé à essayer de faire fonctionner le code de ces deux entraînements avec l'aide de Solène EVAIN et Adrien PUPIERA, deux chercheurs habitués à utiliser de type de code pour le lancement d'entraînement et de fine-tuning. Malgré le fait que nous n'ayons pas trouvé comment résoudre le problème empêchant le fonctionnement de ces entraînements, nous avons au moins trouvé quel était le problème, ce dernier venait du fait que, pour une raison nous échappant, le script python cherche à lancer l'entraînement sur la totalité de l'enregistrement et non extrait par extrait, rendant le calcul beaucoup trop lourd.

3.2 Discussion générale des résultats

Les résultats obtenus lors de ce stage sont encourageants et démontrent la faisabilité de l'entraînement d'un système de reconnaissance automatique de la parole adapté aux enseignants en maternelle.
Le système d'ASR utilisant wav2vec2 pré-entrainé sur Common Voice obtient un taux d'erreur mots (WER) de 29,98% sur les données de test DyLNet. Bien que perfectible, ce résultat n'est pas surprenant étant donné que le modèle n'a pas été entrainé spécifiquement sur des données d'enseignants. Il servait surtout de baseline pour évaluer l'intérêt de l'adaptation.
Le fine-tuning du système sur les données DyLNet permet d'améliorer significativement les performances, avec un WER de 17,89% obtenu après déblocage des couches de ce dernier et 85 epochs d'entraînement. Ainsi, ce résultat valide l'intérêt d'adapter les modèles génériques de reconnaissance vocale aux particularités des données cibles, ici la parole adressée aux enfants en milieu scolaire.

L'analyse des résultats par locuteur et activité montre, quant à elle, que les performances varient sensiblement selon ces paramètres. Cette variabilité peut s'expliquer par les conditions d'enregistrement (bruit ambiant) et les particularités de chaque enseignant (débit de parole, vocabulaire employé, etc). Elle met en évidence la complexité de la tâche de reconnaissance automatique de la parole en milieu scolaire.

Il reste cependant dommage que les expérimentations d'entraînement « from scratch » n'aient pu aboutir, car elles auraient permis de comparer différentes stratégies d'adaptation des systèmes. Néanmoins, le fine-tuning semble déjà apporter un gain substantiel.

En résumé, ce stage a permis de poser les bases d'un système de reconnaissance vocal adapté aux enseignants, et ouvre des perspectives intéressantes pour de futurs travaux d'amélioration. L'approche par fine-tuning semble prometteuse et mériterait d'être explorée plus avant, en augmentant la quantité de données d'entraînement notamment.

Bilan

Ce stage de six mois au sein de l'équipe de recherche en traitement automatique de la parole du LIG restera une expérience particulièrement enrichissante dans mon parcours universitaire. Il m'a permis d'approfondir de façon concrète mes connaissances en reconnaissance automatique de la parole, domaines que nous n'avions abordés que théoriquement lors de ma formation en master Industrie de la Langue. De plus, l'intégration au sein du laboratoire d'informatique de Grenoble a été très stimulante. J'ai pu observer le fonctionnement d'une équipe de recherche et découvrir le monde de la recherche. Les multiples domaines d'expertise représentée dans ce laboratoire et l'investissement des chercheurs dans le développement de nouvelles approches m'ont beaucoup impressionné.
J'ai fortement apprécié le temps consacré à la prise en main des outils SpeechBrain et wav2vec2. Cette phase était indispensable pour bien maîtriser par la suite les différentes fonctionnalités et la façon dont les modèles peuvent être configurés et entraînés. SpeechBrain m'a semblé être un choix pertinent grâce à sa facilité d'utilisation et d'intégration à Python.
Par la suite, le travail de comparaison de différents modèles de wav2vec2 sur les données DyLNet s'est révélé très intéressant. J'ai beaucoup appris en conduisant ces multiples expériences d'entraînement de modèles et en analysant précisément leurs performances respectives. Cependant, bien que très intéressante, cette partie du stage a énormément affecté ma motivation et mon enthousiasme, notamment à cause des bugs répétés de certains modèles et du non-fonctionnement de l'entraînement des deux modèles devant être entraînés sur DyLNet.
Les résultats obtenus restes prometteurs et ouvrent des perspectives intéressantes pour la suite, et ce même si ces derniers peuvent encore être améliorés.

Au cours de ce stage, j'ai pu acquérir de nouvelles compétences techniques en traitement automatique de la parole, que ce soit au niveau de la préparation des données, du choix et de l'entraînement de modèle ou de l'évaluation des performances. J'ai apprécié disposer d'une grande autonomie tout en bénéficiant des conseils avisés de mes encadrants. Grâce à ces derniers, j'ai aussi pu découvrir le monde de la recherche académique, appliquer mes connaissances théoriques à un cas pratique complexe et surtout en apprendre beaucoup plus sur la reconnaissance automatique de la parole. Cette expérience m'a beaucoup apporté sur le plan technique, mais elle m'a aussi énormément apporté sur le plan personnel. Cette dernière m'a conforté dans le fait qu'aujourd'hui, travailler dans le milieu de la recherche n'était pas quelque chose qui m'intéressait, et ce, malgré le fait que le stage ce soit globalement très bien passé. Ce stage m'a aussi confronté à mes problèmes d'organisation et encore plus à mes

problèmes de concentration (encore merci à madame Solange ROSSATO pour ses précieux conseils). Ainsi, ce dernier m'a permis de me rendre compte que, bien que j'aime ce qui touche à l'informatique, au code et au traitement automatique des langues, il était nécessaire que je prenne mes distances avec ce milieu, du moins pour un temps, afin de découvrir d'autres domaines pouvant potentiellement plus me convenir.

Références

Commonvoice fr github repository. https://github.com/common-voice/commonvoice-fr. Projet CommonVoice FR sur GitHub.

Ahmed Ali and Steve Renals. Word error rate estimation for speech recognition : e-wer. pages 20–24, 2018. URL https://www.pure.ed.ac.uk/ws/portalfiles/portal/63902419/ewer_acl2018.pdf.

Rosana Ardila, Megan Branson, Kelly Davis, Michael Kohler, Josh Meyer, Michael Henretty, Reuben Morais, Lindsay Saunders, Francis Tyers, and Gregor Weber. Common voice : A massively-multilingual speech corpus. In *Proceedings of the Twelfth Language Resources and Evaluation Conference*, pages 4218–4222, Marseille, France, May 2020. European Language Resources Association. ISBN 979-10-95546-34-4. URL https://aclanthology.org/2020.lrec-1.520.

Alexei Baevski, Henry Zhou, Abdelrahman Mohamed, and Michael Auli. Wav2vec 2.0 : A Framework for Self-Supervised Learning of Speech Representations. abs/2006.11477, 2021. URL https://proceedings.neurips.cc/paper/2020/file/92d1e1eb1cd6f9fba3227870bb6d7f07-Paper.pdf.

Peter Bell. Automatic speech recognition : Introduction. 2020. URL http://www.inf.ed.ac.uk/teaching/courses/asr/2019-20/asr01-intro.pdf.

Hélène Bouchet, Isabelle Rousset, and Aurélie Nardy. Projet dylnet : développement du langage oral et sociabilité à l'école maternelle. November 2017. URL https://hal.science/hal-01983993âĘŮ.

Paul Deleglise and Carole Lailler. Quel type de systèmes utiliser pour la transcription automatique du français ? les hmm font de la résistance. 1 :154–162, 2020.

Lucile Gelin. *Reconnaissance automatique de la parole d'enfants apprenant.e.s lecteur.ice.s en salle de classe : modélisation acoustique de phonèmes*. PhD thesis, Université Paul Sabatier - Toulouse III, 2022. URL https://theses.hal.science/tel-03715653.

Daniel Jurafsky and James H. Martin. N-gram language models. pages 1–29, 2022. URL https://web.stanford.edu/~jurafsky/slp3/3.pdf.

Hà NGUYEN. Miai collab Dylnet, 2022. Document interne non publié.

OpenAI. Introducing Whisper. URL https://openai.com/research/whisper.

Vassil Panayotov and Daniel Povey. Librispeech asr corpus. Online, 2015. URL https://www.openslr.org/12. Open Speech and Language Resources.

Titouan PARCOLLET. asr-wav2vec2-commonvoice-fr. https://huggingface.co/speechbrain/asr-wav2vec2-commonvoice-fr/tree/main, 2021/2022. Model pré-entraîné de wav2vec2.0 sur les données du corpus CommonVoice.

Mirco Ravanelli, Titouan Parcollet, Peter Plantinga, Aku Rouhe, Samuele Cornell, Loren Lugosch, Cem Subakan, Nauman Dawalatabad, Abdelwahab Heba, Jianyuan Zhong, Ju-Chieh Chou, Sung-Lin Yeh, Szu-Wei Fu, Chien-Feng Liao, Elena Rastorgueva, François Grondin, William Aris, Hwidong Na, Yan Gao, Renato De Mori, and Yoshua Bengio. SpeechBrain : A general-purpose speech toolkit. https://github.com/speechbrain/speechbrain/tree/develop, 2021a. arXiv :2106.04624.

Mirco Ravanelli, Titouan Parcollet, Aku Rouhe, Peter Plantinga, Elena Rastorgueva, Loren Lugosch, Nauman Dawalatabad, Chou Ju-Chieh, Abdel Heba, Francois Grondin, William Aris, Chien-Feng Liao, Samuele Cornell, Sung-Lin Yeh, Hwidong Na, Yan Gao, Szu-Wei Fu, Cem Subakan, Renato De Mori, and Yoshua Bengio. Speechbrain. https://github.com/speechbrain/speechbrain, 2021b.

Evain Solène, Manh Ha Nguyen Hà, Hang Le, Marcely Zanon Boito, Salima Mdhaffar, Sina Alisamir, Ziyi Tong, Natalia Tomashenko, Marco Dinarelli, Titouan Parcollet, et al. Task agnostic and task specific self-supervised learning from speech with LeBenchmark. 2021. URL https://hal.science/hal-03407172.

Annexes

Script python permettant de récupérer les données des fichiers .eaf et script bash de lancement

```
import argparse
import glob
import eaf_reader
import os
from tqdm import tqdm

def get_annotation_list(file_name):
    eaf_file = eaf_reader.Eaf(file_name)
    tier_name = list(filter(lambda x: 'nettoye' in x,
    eaf_file.get_tier_names()))[0]
    annotations_list = ""
    for start_time, end_time, text in               eaf_file.get_annotation_data_fo
    new_line = "{}\t{}\t{}".format(start_time, end_time, text)
        #if 'CVE_REF' in text:
        #   cve_ref = re.search('CVE_REF="([^"]+)"', text).group(1)
        #  cve_value_desc = ''
        # for cv_entry in eaf_file.get_cv_entries():
            #    if cv_entry.get('CVE_ID') == cve_ref:
             #       cve_value_desc = cv_entry.find('CVE_VALUE').get('DESCRIPTION')
              #      break
           # new_line += "\t{}".format(cve_value_desc)
        #annotations_list = annotations_list + new_line + "\n"
    return(annotations_list)

def write_as_txt(source_dir, output_dir):
        '''

        '''
        if not os.path.exists(output_dir):
                os.mkdir(output_dir)
        eaf_files_list = glob.glob('{}/*VB.eaf'.format(source_dir))
        print(eaf_files_list)

        for eaf_file_path in tqdm(eaf_files_list):
```

```
                annotations_list = get_annotation_list(eaf_file_path)
                #txt_file_path = ''.join([output_dir, eaf_file_path[len(source_dir)
                txt_file_path = ''.join([source_dir, eaf_file_path[len(source_dir):
                print(txt_file_path)
                with open(txt_file_path, 'w') as file:
                        file.write(annotations_list)

        print("{} files converted successfully!".format(len(eaf_files_list)))

def get_args():
        '''

        '''
        parser = argparse.ArgumentParser(description='Converts .eaf to .txt')
        parser.add_argument(
                'source_dir',
                type=str,
                help='source directory'
        )
        parser.add_argument(
                'output_dir',
                type=str,
                help='output directory'
        )
        return parser.parse_args()

def main():
        args = get_args()
        write_as_txt(args.source_dir, args.output_dir)

if __name__ == "__main__":
        main()
```

```
#!/bin/bash
```

```
x=$(find P2-Juin-2019/ -name "*VB.eaf")
for i in $x; do
    source_dir=$(dirname $i)
    python3 elan-eaf-reader/scripts/eaf2txt.py $source_dir .
done
```

script python permettant de récupérer les chemins d'accès aux fichiers de transcriptions et script bash de lancement

```
    import os
import sys

# Récupérer le nom du dossier à partir des arguments de la ligne de commande
dossier = sys.argv[1]

# Liste pour stocker les chemins des fichiers .txt
chemins_fichiers_txt = []

# Parcourir le dossier et chercher les fichiers .txt
for nom_fichier in os.listdir(dossier):
    chemin_fichier = os.path.join(dossier, nom_fichier)
    if os.path.isfile(chemin_fichier) and nom_fichier.endswith('.txt'):
        chemins_fichiers_txt.append(chemin_fichier)

# Écrire les chemins des fichiers .txt à la fin du fichier 'Train.txt'
with open('Test.txt', 'a') as fichier_chemins:
    for chemin in chemins_fichiers_txt:
        fichier_chemins.write(chemin + '\n')
```

```
    #!/bin/bash
        x=$(find partitions/Test_files/ -name "*_VB.txt")

        for i in $x; do
            source_dir=$(dirname $i)

            python3 recupchemin.py $source_dir .
```

```
done
```

Scripts python permettant de réaliser le fichier JSON

```
import json
import unicodedata
import re
from collections import OrderedDict
from os.path import exists

data = OrderedDict()
accented_letters = True

def read_file(input_file, adult=True):
    """
    Parse eaf input_file then normalize the text
    """
    id_ = input_file.split('/')[-1].split('.')[0]
    print("read_file ok")

    with open(input_file, 'r') as fr:
    for i, line in enumerate(fr):
        wav = input_file.replace('transcriptions', 'audios').replace('_VB.txt',
        if not exists(wav):
            continue

        # Get important info of a segment (start, end, text)
        start = line.split('\t')[0]
        end = line.split('\t')[1]
        #words = line.replace(start + '\t' + end + '\t', '').strip()
        act = line.split('\t')[2]
        words = line.split('\t')[3]

        # from SB recipe:
        words = unicode_normalisation(words)
        words = re.sub(
            "[^''A-Za-z0-9À-ÖØ-öø-ÿ-eæœaceaui]+", " ", words
        ).upper()
        words = words.replace("'", " ")
```

```
            words = words.replace("'", " ")

            # Remove accents if specified
            if not accented_letters:
                words = strip_accents(words)
                words = words.replace("'", " ")
                words = words.replace("'", " ")

            # Remove multiple spaces
            words = re.sub(" +", " ", words)

            # Remove spaces at the beginning and the end of the sentence
            words = words.lstrip().rstrip()

            # Skip if a segment has no text
            if words == '':
                continue

            # Getting chars
            chars = words.replace(" ", "_")
            chars = " ".join([char for char in chars][:])

            seg_id = id_ + '_' + str(i+1)
            seg_id = id_ + '_' + str(start) + '_' + str(end)
            data[seg_id] = {}
            data[seg_id]['spk_id'] = id_
            data[seg_id]['start_seg'] = float(start) / 1000
            data[seg_id]['end_seg'] = float(end) / 1000
            data[seg_id]['duration'] = float((float(end) - float(start)) / 1000)
            data[seg_id]['wav'] = wav
            data[seg_id]['activité'] = act
            data[seg_id]['wrd'] = words
            data[seg_id]['char'] = chars
            print("read_file ok 2")

def unicode_normalisation(text):
    try:
        text = unicode(text, "utf-8")
    except NameError:  # unicode is a default on python 3
        pass
```

```
    return str(text)


def strip_accents(text):

    text = (
        unicodedata.normalize("NFD", text)
        .encode("ascii", "ignore")
        .decode("utf-8")
    )
    print("strip_accents ok")
    return str(text)

def main():
    with open('partitions/Train.txt', 'r') as fr:
        for line in fr:
            read_file(line.strip(), adult=True)
    with open('Train.json', 'w') as outfile:
        json.dump(data, outfile, indent=4, ensure_ascii=False)
        print("main ok 2")


if __name__ == main():
    main()
```

Extrait du code YAML (hyperparamètres) pour le fine-tuning sur les données de DyLNet du modèle Wav2vec2-FR-7K-large entraîné sur Common Voice FR avec wav2vec freeze

```
# ################################
# Model: wav2vec2 + DNN + CTC
# Augmentation: SpecAugment
# Authors: Titouan Parcollet 2021
# ################################

# Seed needs to be set at top of yaml, before objects with parameters are made
seed: 1234
__set_seed: !!python/object/apply:torch.manual_seed [!ref <seed>]
output_folder: !ref TRAIN/wav2vec_CV+dylnet/results/<seed>
```

```
wer_file: !ref <output_folder>/wer.txt
save_folder: !ref <output_folder>/save
train_log: !ref <output_folder>/train_log.txt

# URL for the biggest LeBenchmark wav2vec french.
pretrain_path: speechbrain/asr-wav2vec2-commonvoice-fr
wav2vec2_hub: LeBenchmark/wav2vec2-FR-7K-large

# Data files
data_folder: partitions  # e.g, /localscratch/cv-corpus-5.1-2020-06-22/fr
accented_letters: True
language: fr # use 'it' for Italian, 'rw' for Kinyarwanda, 'en' for english
train_json: !ref <data_folder>/Train.json
valid_json: !ref <data_folder>/Dev.json
test_json: !ref <data_folder>/Test.json
skip_prep: True # Skip data preparation

# We remove utterance slonger than 10s in the train/dev/test sets as
# longer sentences certainly correspond to "open microphones".
avoid_if_longer_than: 30.0
avoid_if_smaller_than: 0.0

# Training parameters
number_of_epochs: 49
number_of_ctc_epochs: 49
lr: 1.0
lr_wav2vec: 1
ctc_weight: 0.3
sorting: ascending
auto_mix_prec: False
sample_rate: 16000
ckpt_interval_minutes: 30 # save checkpoint every N min

# With data_parallel batch_size is split into N jobs
# With DDP batch_size is multiplied by N jobs
# Must be 6 per GPU to fit 16GB of VRAM
batch_size: 4
test_batch_size: 4

dataloader_options:
```

```
    batch_size: !ref <batch_size>
    num_workers: 8
test_dataloader_options:
    batch_size: !ref <test_batch_size>
    num_workers: 8

# BPE parameters
token_type: char  # ["unigram", "bpe", "char"]
character_coverage: 1.0

tokenizer: !new:sentencepiece.SentencePieceProcessor
# Model parameters
activation: !name:torch.nn.LeakyReLU
wav2vec_output_dim: 1024
dnn_neurons: 1024
freeze_wav2vec: True
```

Extrait du code YAML (hyperparamètres) pour le fine-tuning sur les données de DyLNet du modèle Wav2vec2-FR-7K-large entraîné sur Common Voice FR modèle avec wav2vec defreeze

```
    # ################################
# Model: wav2vec2 + DNN + CTC
# Augmentation: SpecAugment
# Authors: Titouan Parcollet 2021
# ################################

# Seed needs to be set at top of yaml, before objects with parameters are made
seed: 1234
__set_seed: !!python/object/apply:torch.manual_seed [!ref <seed>]
output_folder: !ref TRAIN/wav2vec_CV+dylnet+defreeze/results/<seed>
wer_file: !ref <output_folder>/wer.txt
save_folder: !ref <output_folder>/save
train_log: !ref <output_folder>/train_log.txt

# URL for the biggest LeBenchmark wav2vec french.
pretrain_path: speechbrain/asr-wav2vec2-commonvoice-fr
wav2vec2_hub: LeBenchmark/wav2vec2-FR-7K-large
```

```
# Data files
data_folder: partitions  # e.g, /localscratch/cv-corpus-5.1-2020-06-22/fr
accented_letters: True
language: fr # use 'it' for Italian, 'rw' for Kinyarwanda, 'en' for english
train_json: !ref <data_folder>/Train.json
valid_json: !ref <data_folder>/Dev.json
test_json: !ref <data_folder>/Test.json
skip_prep: True # Skip data preparation

# We remove utterance slonger than 10s in the train/dev/test sets as
# longer sentences certainly correspond to "open microphones".
avoid_if_longer_than: 30.0
avoid_if_smaller_than: 0.0

# Training parameters
number_of_epochs: 150 #30
number_of_ctc_epochs: 150
lr: 1.0
lr_wav2vec: 1.0
ctc_weight: 0.3
sorting: ascending
auto_mix_prec: False
sample_rate: 16000
ckpt_interval_minutes: 30 # save checkpoint every N min

# With data_parallel batch_size is split into N jobs
# With DDP batch_size is multiplied by N jobs
# Must be 6 per GPU to fit 16GB of VRAM
batch_size: 5
test_batch_size: 5

dataloader_options:
    batch_size: !ref <batch_size>
    num_workers: 6
test_dataloader_options:
    batch_size: !ref <test_batch_size>
    num_workers: 6

# BPE parameters
token_type: char  # ["unigram", "bpe", "char"]
```

```
character_coverage: 1.0

tokenizer: !new:sentencepiece.SentencePieceProcessor
# Model parameters
activation: !name:torch.nn.LeakyReLU
wav2vec_output_dim: 1024
dnn_neurons: 1024
freeze_wav2vec: False
```

Extrait du code YAML (hyperparamètres) pour l'entraînement sur les données de DyLNet du modèle Wav2vec2-FR-7K-large avec wav2vec freeze

```
    # ################################
# Model: wav2vec2 + DNN + CTC
# Augmentation: SpecAugment
# Authors: Titouan Parcollet 2021
# ################################

# Seed needs to be set at top of yaml, before objects with parameters are made
seed: 1234
__set_seed: !!python/object/apply:torch.manual_seed [!ref <seed>]
output_folder: !ref TRAIN/wav2vec+dylnet/Results/<seed>
wer_file: !ref <output_folder>/wer.txt
save_folder: !ref <output_folder>/save
train_log: !ref <output_folder>/train_log.txt

# URL for the biggest LeBenchmark wav2vec french.
wav2vec2_hub: LeBenchmark/wav2vec2-FR-7K-base
wav2vec2_folder: !ref <save_folder>/wav2vec2_checkpoint

# Data files
data_folder: partitions/mini  # e.g, /localscratch/cv-corpus-5.1-2020-06-22/fr
train_json: !ref <data_folder>/Train.json
valid_json: !ref <data_folder>/Dev.json
test_json: !ref <data_folder>/Test.json
accented_letters: True
language: fr # use 'it' for Italian, 'rw' for Kinyarwanda, 'en' for english
skip_prep: False # Skip data preparation
```

```
# We remove utterance slonger than 10s in the train/dev/test sets as
# longer sentences certainly correspond to "open microphones".
avoid_if_longer_than: 6.0
avoid_if_smaller_than: 1

# Training parameters
number_of_epochs: 30
number_of_ctc_epochs: 30
lr: 0.001
lr_wav2vec: 0.0001
ctc_weight: 0.03
sorting: ascending
auto_mix_prec: False
sample_rate: 16000
ckpt_interval_minutes: 30 # save checkpoint every N min

# With data_parallel batch_size is split into N jobs
# With DDP batch_size is multiplied by N jobs
# Must be 6 per GPU to fit 16GB of VRAM
batch_size: 1
test_batch_size: 1

dataloader_options:
    batch_size: !ref <batch_size>
    num_workers: 1
test_dataloader_options:
    batch_size: !ref <test_batch_size>
    num_workers: 1

# BPE parameters
token_type: char  # ["unigram", "bpe", "char"]
character_coverage: 1.0

# Model parameters
#activation: !name:torch.nn.LeakyReLU
wav2vec_output_dim: 1024
dnn_neurons: 1024
freeze_wav2vec: True
```

Code python pour le lancement des fine-tuning (freeze ou non) sur les données de DyLNet du modèle Wav2vec2-FR-7K-large entraîné sur Common Voice FR

```
    #!/usr/bin/env python3
import sys
import torch
import logging
import speechbrain as sb
import torchaudio
from hyperpyyaml import load_hyperpyyaml
from speechbrain.tokenizers.SentencePiece import SentencePiece
from speechbrain.utils.data_utils import undo_padding
from speechbrain.utils.distributed import run_on_main
from speechbrain.utils.parameter_transfer import Pretrainer
from sentencepiece import SentencePieceProcessor

"""Recipe for training a sequence-to-sequence ASR system with CommonVoice.
The system employs a wav2vec2 encoder and a CTC decoder.
Decoding is performed with greedy decoding (will be extended to beam search).

To run this recipe, do the following:
> python train_with_wav2vec2.py hparams/train_with_wav2vec2.yaml

With the default hyperparameters, the system employs a pretrained wav2vec2 encoder.
The wav2vec2 model is pretrained following the model given in the hprams file.
It may be dependent on the language.

The neural network is trained with CTC on sub-word units estimated with
Byte Pairwise Encoding (BPE).

The experiment file is flexible enough to support a large variety of
different systems. By properly changing the parameter files, you can try
different encoders, decoders, tokens (e.g, characters instead of BPE),
training languages (all CommonVoice languages), and many
other possible variations.

Authors
 * Titouan Parcollet 2021
"""

logger = logging.getLogger(__name__)
```

```
# Define training procedure
class ASR(sb.core.Brain):
    def compute_forward(self, batch, stage):
        """Forward computations from the waveform batches to the output probabiliti

        batch = batch.to(self.device)
        wavs, wav_lens = batch.sig
        tokens_bos, _ = batch.tokens_bos
        wavs, wav_lens = wavs.to(self.device), wav_lens.to(self.device)

        if stage == sb.Stage.TRAIN:
            if hasattr(self.hparams, "augmentation"):
                wavs = self.hparams.augmentation(wavs, wav_lens)

        # Forward pass
        feats = self.modules.wav2vec2(wavs)
        x = self.modules.enc(feats)
        logits = self.modules.ctc_lin(x)
        p_ctc = self.hparams.log_softmax(logits)

        return p_ctc, wav_lens

    def compute_objectives(self, predictions, batch, stage):
        """Computes the loss (CTC) given predictions and targets."""

        p_ctc, wav_lens = predictions

        ids = batch.id
        tokens_eos, tokens_eos_lens = batch.tokens_eos
        tokens, tokens_lens = batch.tokens

        loss = self.hparams.ctc_cost(p_ctc, tokens, wav_lens, tokens_lens)

        if stage != sb.Stage.TRAIN:
            # Decode token terms to words
            sequence = sb.decoders.ctc_greedy_decode(
                p_ctc, wav_lens, blank_id=self.hparams.blank_index
            )
```

```
            predicted_words = [self.tokenizer.decode_ids(utt_seq).split(" ") for ut

            # Convert indices to words
            target_words = undo_padding(tokens, tokens_lens)
            target_words = [self.tokenizer.decode_ids(utt_seq).split(" ") for utt_s

            self.wer_metric.append(ids, predicted_words, target_words)
            self.cer_metric.append(ids, predicted_words, target_words)

        return loss

    def fit_batch(self, batch):
        """Train the parameters given a single batch in input"""
        if self.auto_mix_prec:

            if not self.hparams.wav2vec2.freeze:
                self.wav2vec_optimizer.zero_grad()
            self.model_optimizer.zero_grad()

            with torch.cuda.amp.autocast():
                outputs = self.compute_forward(batch, sb.Stage.TRAIN)
                loss = self.compute_objectives(outputs, batch, sb.Stage.TRAIN)

            self.scaler.scale(loss).backward()
            if not self.hparams.wav2vec2.freeze:
                self.scaler.unscale_(self.wav2vec_optimizer)
            self.scaler.unscale_(self.model_optimizer)

            if self.check_gradients(loss):
                if not self.hparams.wav2vec2.freeze:
                    self.scaler.step(self.wav2vec_optimizer)
                self.scaler.step(self.model_optimizer)

            self.scaler.update()
        else:
            outputs = self.compute_forward(batch, sb.Stage.TRAIN)

            loss = self.compute_objectives(outputs, batch, sb.Stage.TRAIN)
            loss.backward()
```

```
            if self.check_gradients(loss):
                if not self.hparams.wav2vec2.freeze:
                    self.wav2vec_optimizer.step()
                self.model_optimizer.step()

            if not self.hparams.wav2vec2.freeze:
                self.wav2vec_optimizer.zero_grad()
            self.model_optimizer.zero_grad()

        return loss.detach()

    def evaluate_batch(self, batch, stage):
        """Computations needed for validation/test batches"""
        predictions = self.compute_forward(batch, stage=stage)
        with torch.no_grad():
            loss = self.compute_objectives(predictions, batch, stage=stage)
        return loss.detach()

    def on_stage_start(self, stage, epoch):
        """Gets called at the beginning of each epoch"""
        if stage != sb.Stage.TRAIN:
            self.cer_metric = self.hparams.cer_computer()
            self.wer_metric = self.hparams.error_rate_computer()

    def on_stage_end(self, stage, stage_loss, epoch):
        """Gets called at the end of an epoch."""
        # Compute/store important stats
        stage_stats = {"loss": stage_loss}
        if stage == sb.Stage.TRAIN:
            self.train_stats = stage_stats
        else:
            stage_stats["CER"] = self.cer_metric.summarize("error_rate")
            stage_stats["WER"] = self.wer_metric.summarize("error_rate")

        # Perform end-of-iteration things, like annealing, logging, etc.
        if stage == sb.Stage.VALID:
            old_lr_model, new_lr_model = self.hparams.lr_annealing_model(
                stage_stats["loss"]
            )
```

```
            old_lr_wav2vec, new_lr_wav2vec = self.hparams.lr_annealing_wav2vec(
                stage_stats["loss"]
            )
            sb.nnet.schedulers.update_learning_rate(
                self.model_optimizer, new_lr_model
            )
            if not self.hparams.wav2vec2.freeze:
                sb.nnet.schedulers.update_learning_rate(
                    self.wav2vec_optimizer, new_lr_wav2vec
                )
            self.hparams.train_logger.log_stats(
                stats_meta={
                    "epoch": epoch,
                    "lr_model": old_lr_model,
                    "lr_wav2vec": old_lr_wav2vec,
                },
                train_stats=self.train_stats,
                valid_stats=stage_stats,
            )
            self.checkpointer.save_and_keep_only(
                meta={"WER": stage_stats["WER"]}, min_keys=["WER"],
            )
        elif stage == sb.Stage.TEST:
            self.hparams.train_logger.log_stats(
                stats_meta={"Epoch loaded": self.hparams.epoch_counter.current},
                test_stats=stage_stats,
            )
            with open(self.hparams.wer_file, "w") as w:
                self.wer_metric.write_stats(w)

    def init_optimizers(self):
        "Initializes the wav2vec2 optimizer and model optimizer"

        # If the wav2vec encoder is unfrozen, we create the optimizer
        if not self.hparams.wav2vec2.freeze:
            self.wav2vec_optimizer = self.hparams.wav2vec_opt_class(
                self.modules.wav2vec2.parameters()
            )
            if self.checkpointer is not None:
                self.checkpointer.add_recoverable(
```

```
                    "wav2vec_opt", self.wav2vec_optimizer
                )

        self.model_optimizer = self.hparams.model_opt_class(
            self.hparams.model.parameters()
        )

        if self.checkpointer is not None:
            self.checkpointer.add_recoverable("modelopt", self.model_optimizer)


# Define custom data procedure
def dataio_prepare(hparams, tokenizer):
    """This function prepares the datasets to be used in the brain class.
    It also defines the data processing pipeline through user-defined functions."""

    # 1. Define datasets
    data_folder = hparams["data_folder"]

    train_data = sb.dataio.dataset.DynamicItemDataset.from_json(
        json_path=hparams["train_json"], replacements={"data_root": data_folder},
    )

    if hparams["sorting"] == "ascending":
        # we sort training data to speed up training and get better results.
        train_data = train_data.filtered_sorted(
            sort_key="duration",
            key_max_value={"duration": hparams["avoid_if_longer_than"]},
            key_min_value={"duration": hparams["avoid_if_smaller_than"]},
        )
        # when sorting do not shuffle in dataloader ! otherwise is pointless
        hparams["dataloader_options"]["shuffle"] = False

    elif hparams["sorting"] == "descending":
        train_data = train_data.filtered_sorted(
            sort_key="duration",
            reverse=True,
            key_max_value={"duration": hparams["avoid_if_longer_than"]},
            key_min_value={"duration": hparams["avoid_if_smaller_than"]},
        )
```

```
    # when sorting do not shuffle in dataloader ! otherwise is pointless
    hparams["dataloader_options"]["shuffle"] = False

elif hparams["sorting"] == "random":
    pass

else:
    raise NotImplementedError(
        "sorting must be random, ascending or descending"
    )

valid_data = sb.dataio.dataset.DynamicItemDataset.from_json(
    json_path=hparams["valid_json"], replacements={"data_root": data_folder},
)
# We also sort the validation data so it is faster to validate

valid_data = valid_data.filtered_sorted(
    sort_key="duration",
    reverse=True,
    key_max_value={"duration": hparams["avoid_if_longer_than"]},
    key_min_value={"duration": hparams["avoid_if_smaller_than"]},
)

test_data = sb.dataio.dataset.DynamicItemDataset.from_json(
    json_path=hparams["test_json"], replacements={"data_root": data_folder},
)
# We also sort the validation data so it is faster to validate
test_data = test_data.filtered_sorted(
    sort_key="duration",
    reverse=True,
    key_max_value={"duration": hparams["avoid_if_longer_than"]},
    key_min_value={"duration": hparams["avoid_if_smaller_than"]},
)

datasets = [train_data, valid_data, test_data]

# 2. Define audio pipeline:
@sb.utils.data_pipeline.takes("wav", "start_seg", "end_seg")
@sb.utils.data_pipeline.provides("sig")
def audio_pipeline(wav, start_seg, end_seg):
```

```
        info = torchaudio.info(wav)
        start = int(float(start_seg) * info.sample_rate)
        stop = int(float(end_seg) * info.sample_rate)
        speech_segment = {"file" : wav, "start" : start, "stop" : stop}
        sig = sb.dataio.dataio.read_audio(speech_segment)
        if info.num_channels > 1:
            sig = torch.mean(sig, dim=1)
        resampled = torchaudio.transforms.Resample(
            info.sample_rate, hparams["sample_rate"],
        )(sig)
        return resampled

    sb.dataio.dataset.add_dynamic_item(datasets, audio_pipeline)

    # 3. Define text pipeline:
    @sb.utils.data_pipeline.takes("wrd")
    @sb.utils.data_pipeline.provides(
        "tokens_list", "tokens_bos", "tokens_eos", "tokens"
    )
    def text_pipeline(wrd):
        #tokens_list = tokenizer.sp.encode_as_ids(wrd)
        tokens_list = tokenizer.encode_as_ids(wrd)
        yield tokens_list
        tokens_bos = torch.LongTensor([hparams["bos_index"]] + (tokens_list))
        yield tokens_bos
        tokens_eos = torch.LongTensor(tokens_list + [hparams["eos_index"]])
        yield tokens_eos
        tokens = torch.LongTensor(tokens_list)
        yield tokens

    sb.dataio.dataset.add_dynamic_item(datasets, text_pipeline)

    # 4. Set output:
    sb.dataio.dataset.set_output_keys(
        datasets, ["id", "sig", "tokens_bos", "tokens_eos", "tokens"],
    )
    return train_data, valid_data, test_data


if __name__ == "__main__":
```

```
# Load hyperparameters file with command-line overrides
hparams_file, run_opts, overrides = sb.parse_arguments(sys.argv[1:])
with open(hparams_file) as fin:
    hparams = load_hyperpyyaml(fin, overrides)

# If distributed_launch=True then
# create ddp_group with the right communication protocol
sb.utils.distributed.ddp_init_group(run_opts)

# Dataset preparation (parsing CommonVoice)
#from common_voice_prepare import prepare_common_voice  # noqa

# Create experiment directory
sb.create_experiment_directory(
    experiment_directory=hparams["output_folder"],
    hyperparams_to_save=hparams_file,
    overrides=overrides,
)

# Due to DDP, we do the preparation ONLY on the main python process
#run_on_main(
#    prepare_common_voice,
#    kwargs={
#        "data_folder": hparams["data_folder"],
#        "save_folder": hparams["save_folder"],
#        "train_tsv_file": hparams["train_tsv_file"],
#        "dev_tsv_file": hparams["dev_tsv_file"],
#        "test_tsv_file": hparams["test_tsv_file"],
#        "accented_letters": hparams["accented_letters"],
#        "language": hparams["language"],
#        "skip_prep": hparams["skip_prep"],
#    },
#)


run_on_main(hparams["pretrainer"].collect_files)
hparams["pretrainer"].load_collected(device=run_opts["device"])

# Trainer initialization
```

```
asr_brain = ASR(
    modules=hparams["modules"],
    hparams=hparams,
    run_opts=run_opts,
    checkpointer=hparams["checkpointer"],
)

# Adding objects to trainer.
#asr_brain.tokenizer = tokenizer
asr_brain.tokenizer = hparams["tokenizer"]

# Create the datasets objects as well as tokenization and encoding :-D
train_data, valid_data, test_data = dataio_prepare(hparams, asr_brain.tokenizer

# Training
asr_brain.fit(
    asr_brain.hparams.epoch_counter,
    train_data,
    valid_data,
    train_loader_kwargs=hparams["dataloader_options"],
    valid_loader_kwargs=hparams["test_dataloader_options"],
)

# Test
asr_brain.hparams.wer_file = hparams["output_folder"] + "/Test.txt"
asr_brain.evaluate(
    test_data,
    min_key="WER",
    test_loader_kwargs=hparams["test_dataloader_options"],
)

asr_brain.hparams.wer_file = hparams["output_folder"] + "/Train.txt"
asr_brain.evaluate(
    train_data,
    min_key="WER",
    test_loader_kwargs=hparams["test_dataloader_options"],
)

asr_brain.hparams.wer_file = hparams["output_folder"] + "/Valid.txt"
asr_brain.evaluate(
```

```
        valid_data,
        min_key="WER",
        test_loader_kwargs=hparams["test_dataloader_options"],
    )
```

Extrait du code YAML (hyperparamètres) pour l'entraînement sur les données de DyLNet du modèle Wav2vec2-FR-7K-large avec wav2vec defreeze

```
    # ################################
# Model: wav2vec2 + DNN + CTC
# Augmentation: SpecAugment
# Authors: Titouan Parcollet 2021
# ################################

# Seed needs to be set at top of yaml, before objects with parameters are made
seed: 1234
__set_seed: !!python/object/apply:torch.manual_seed [!ref <seed>]
output_folder: !ref TRAIN/wav2vec+dylnet+defreeze/Results/<seed>
wer_file: !ref <output_folder>/wer.txt
save_folder: !ref <output_folder>/save
train_log: !ref <output_folder>/train_log.txt

# URL for the biggest LeBenchmark wav2vec french.
pretrain_path: speechbrain/asr-wav2vec2-commonvoice-fr
wav2vec2_hub: LeBenchmark/wav2vec2-FR-7K-large

# Data files
data_folder: partitions  # e.g, /localscratch/cv-corpus-5.1-2020-06-22/fr
accented_letters: True
language: fr # use 'it' for Italian, 'rw' for Kinyarwanda, 'en' for english
train_json: !ref <data_folder>/Train.json
valid_json: !ref <data_folder>/Dev.json
test_json: !ref <data_folder>/Test.json
skip_prep: True # Skip data preparation

# We remove utterance slonger than 10s in the train/dev/test sets as
# longer sentences certainly correspond to "open microphones".
avoid_if_longer_than: 30.0
avoid_if_smaller_than: 0.0
```

```
# Training parameters
number_of_epochs: 10 #30
number_of_ctc_epochs: 15
lr: 1.0
lr_wav2vec: 0.0001
ctc_weight: 0.3
sorting: ascending
auto_mix_prec: False
sample_rate: 16000
ckpt_interval_minutes: 30 # save checkpoint every N min

# With data_parallel batch_size is split into N jobs
# With DDP batch_size is multiplied by N jobs
# Must be 6 per GPU to fit 16GB of VRAM
batch_size: 1
test_batch_size: 1

dataloader_options:
    batch_size: !ref <batch_size>
    num_workers: 6
test_dataloader_options:
    batch_size: !ref <test_batch_size>
    num_workers: 6

# BPE parameters
token_type: char  # ["unigram", "bpe", "char"]
character_coverage: 1.0

tokenizer: !new:sentencepiece.SentencePieceProcessor
# Model parameters
activation: !name:torch.nn.LeakyReLU
wav2vec_output_dim: 1024
dnn_neurons: 1024
freeze_wav2vec: False
```

Code python pour le lancement des entraînements (freeze ou non) sur les données de DyLNet du modèle Wav2vec2-FR-7K-large

```
    #!/usr/bin/env python3
"""Recipe for training a sequence-to-sequence ASR system with CommonVoice.
The system employs a wav2vec2 encoder and a CTC decoder.
Decoding is performed with greedy decoding (will be extended to beam search).

To run this recipe, do the following:
> python train_with_wav2vec2.py hparams/train_with_wav2vec2.yaml

With the default hyperparameters, the system employs a pretrained wav2vec2 encoder.
The wav2vec2 model is pretrained following the model given in the hprams file.
It may be dependent on the language.

The neural network is trained with CTC on sub-word units estimated with
Byte Pairwise Encoding (BPE).

The experiment file is flexible enough to support a large variety of
different systems. By properly changing the parameter files, you can try
different encoders, decoders, tokens (e.g, characters instead of BPE),
training languages (all CommonVoice languages), and many
other possible variations.

Authors
 * Titouan Parcollet 2021
"""

import sys
import torch
import logging
import speechbrain as sb
import torchaudio
from hyperpyyaml import load_hyperpyyaml
from speechbrain.tokenizers.SentencePiece import SentencePiece
from speechbrain.utils.data_utils import undo_padding
from speechbrain.utils.distributed import run_on_main, if_main_process

logger = logging.getLogger(__name__)


# Define training procedure
class ASR(sb.core.Brain):
```

```
def compute_forward(self, batch, stage):
    """Forward computations from the waveform batches to the output probabiliti

    batch = batch.to(self.device)
    wavs, wav_lens = batch.sig
    tokens_bos, _ = batch.tokens_bos
    wavs, wav_lens = wavs.to(self.device), wav_lens.to(self.device)
    print(wavs.shape)

    if stage == sb.Stage.TRAIN:
        if hasattr(self.hparams, "augmentation"):
            wavs = self.hparams.augmentation(wavs, wav_lens)

    # Forward pass
    feats = self.modules.wav2vec2(wavs, wav_lens)
    x = self.modules.enc(feats)
    logits = self.modules.ctc_lin(x)
    p_ctc = self.hparams.log_softmax(logits)

    return p_ctc, wav_lens


def compute_objectives(self, predictions, batch, stage):
    """Computes the loss (CTC) given predictions and targets."""

    p_ctc, wav_lens = predictions

    ids = batch.id
    tokens_eos, tokens_eos_lens = batch.tokens_eos
    tokens, tokens_lens = batch.tokens

    loss = self.hparams.ctc_cost(p_ctc, tokens, wav_lens, tokens_lens)

    if stage != sb.Stage.TRAIN:
        # Decode token terms to words
        sequence = sb.decoders.ctc_greedy_decode(
            p_ctc, wav_lens, blank_id=self.hparams.blank_index
        )

        predicted_words = self.tokenizer(sequence, task="decode_from_list")
```

```
            # Convert indices to words
            target_words = undo_padding(tokens, tokens_lens)
            target_words = self.tokenizer(target_words, task="decode_from_list")

            self.wer_metric.append(ids, predicted_words, target_words)
            self.cer_metric.append(ids, predicted_words, target_words)

        return loss

    def fit_batch(self, batch):
        """Train the parameters given a single batch in input"""
        should_step = self.step % self.grad_accumulation_factor == 0
        # Managing automatic mixed precision
        # TOFIX: CTC fine-tuning currently is unstable
        # This is certainly due to CTC being done in fp16 instead of fp32
        if self.auto_mix_prec:
            with torch.cuda.amp.autocast():
                with self.no_sync():
                    outputs = self.compute_forward(batch, sb.Stage.TRAIN)
                loss = self.compute_objectives(outputs, batch, sb.Stage.TRAIN)
            with self.no_sync(not should_step):
                self.scaler.scale(
                    loss / self.grad_accumulation_factor
                ).backward()
            if should_step:

                if not self.hparams.wav2vec2.freeze:
                    self.scaler.unscale_(self.wav2vec_optimizer)
                self.scaler.unscale_(self.model_optimizer)
                if self.check_gradients(loss):
                    if not self.hparams.wav2vec2.freeze:
                        if self.optimizer_step >= self.hparams.warmup_steps:
                            self.scaler.step(self.wav2vec_optimizer)
                    self.scaler.step(self.model_optimizer)
                self.scaler.update()
                self.zero_grad()
                self.optimizer_step += 1
        else:
            # This is mandatory because HF models have a weird behavior with DDP
```

```
            # on the forward pass
            with self.no_sync():
                outputs = self.compute_forward(batch, sb.Stage.TRAIN)

            loss = self.compute_objectives(outputs, batch, sb.Stage.TRAIN)

            with self.no_sync(not should_step):
                (loss / self.grad_accumulation_factor).backward()
            if should_step:
                if self.check_gradients(loss):
                    if not self.hparams.wav2vec2.freeze:
                        if self.optimizer_step >= self.hparams.warmup_steps:
                            self.wav2vec_optimizer.step()
                    self.model_optimizer.step()
                self.zero_grad()
                self.optimizer_step += 1

        self.on_fit_batch_end(batch, outputs, loss, should_step)
        return loss.detach().cpu()

    def evaluate_batch(self, batch, stage):
        """Computations needed for validation/test batches"""
        predictions = self.compute_forward(batch, stage=stage)
        with torch.no_grad():
            loss = self.compute_objectives(predictions, batch, stage=stage)
        return loss.detach()

    def on_stage_start(self, stage, epoch):
        """Gets called at the beginning of each epoch"""
        if stage != sb.Stage.TRAIN:
            self.cer_metric = self.hparams.cer_computer()
            self.wer_metric = self.hparams.error_rate_computer()

    def on_stage_end(self, stage, stage_loss, epoch):
        """Gets called at the end of an epoch."""
        # Compute/store important stats
        stage_stats = {"loss": stage_loss}
        if stage == sb.Stage.TRAIN:
            self.train_stats = stage_stats
        else:
```

```
            stage_stats["CER"] = self.cer_metric.summarize("error_rate")
            stage_stats["WER"] = self.wer_metric.summarize("error_rate")

        # Perform end-of-iteration things, like annealing, logging, etc.
        if stage == sb.Stage.VALID:
            old_lr_model, new_lr_model = self.hparams.lr_annealing_model(
                stage_stats["loss"]
            )
            old_lr_wav2vec, new_lr_wav2vec = self.hparams.lr_annealing_wav2vec(
                stage_stats["loss"]
            )
            sb.nnet.schedulers.update_learning_rate(
                self.model_optimizer, new_lr_model
            )
            if not self.hparams.wav2vec2.freeze:
                sb.nnet.schedulers.update_learning_rate(
                    self.wav2vec_optimizer, new_lr_wav2vec
                )
            self.hparams.train_logger.log_stats(
                stats_meta={
                    "epoch": epoch,
                    "lr_model": old_lr_model,
                    "lr_wav2vec": old_lr_wav2vec,
                },
                train_stats=self.train_stats,
                valid_stats=stage_stats,
            )
            self.checkpointer.save_and_keep_only(
                meta={"WER": stage_stats["WER"]}, min_keys=["WER"],
            )
        elif stage == sb.Stage.TEST:
            self.hparams.train_logger.log_stats(
                stats_meta={"Epoch loaded": self.hparams.epoch_counter.current},
                test_stats=stage_stats,
            )
            if if_main_process():
                with open(self.hparams.test_wer_file, "w") as w:
                    self.wer_metric.write_stats(w)

    def init_optimizers(self):
```

```
        "Initializes the wav2vec2 optimizer and model optimizer"

        # If the wav2vec encoder is unfrozen, we create the optimizer
        if not self.hparams.wav2vec2.freeze:
            self.wav2vec_optimizer = self.hparams.wav2vec_opt_class(
                self.modules.wav2vec2.parameters()
            )
            if self.checkpointer is not None:
                self.checkpointer.add_recoverable(
                    "wav2vec_opt", self.wav2vec_optimizer
                )

        self.model_optimizer = self.hparams.model_opt_class(
            self.hparams.model.parameters()
        )

        if self.checkpointer is not None:
            self.checkpointer.add_recoverable("modelopt", self.model_optimizer)

    def zero_grad(self, set_to_none=False):
        if not self.hparams.wav2vec2.freeze:
            self.wav2vec_optimizer.zero_grad(set_to_none)
        self.model_optimizer.zero_grad(set_to_none)


# Define custom data procedure
def dataio_prepare(hparams, tokenizer):
    """This function prepares the datasets to be used in the brain class.
    It also defines the data processing pipeline through user-defined functions."""

    # 1. Define datasets
    data_folder = hparams["data_folder"]

    train_data = sb.dataio.dataset.DynamicItemDataset.from_json(
        json_path=hparams["train_json"], replacements={"data_root": data_folder},
    )

    if hparams["sorting"] == "ascending":
        # we sort training data to speed up training and get better results.
        train_data = train_data.filtered_sorted(
```

```
        sort_key="duration",
        key_max_value={"duration": hparams["avoid_if_longer_than"]},
        key_min_value={"duration": hparams["avoid_if_smaller_than"]},
    )
    # when sorting do not shuffle in dataloader ! otherwise is pointless
    hparams["dataloader_options"]["shuffle"] = False

elif hparams["sorting"] == "descending":
    train_data = train_data.filtered_sorted(
        sort_key="duration",
        reverse=True,
        key_max_value={"duration": hparams["avoid_if_longer_than"]},
        key_min_value={"duration": hparams["avoid_if_smaller_than"]},
    )
    # when sorting do not shuffle in dataloader ! otherwise is pointless
    hparams["dataloader_options"]["shuffle"] = False

elif hparams["sorting"] == "random":
    pass

else:
    raise NotImplementedError(
        "sorting must be random, ascending or descending"
    )

valid_data = sb.dataio.dataset.DynamicItemDataset.from_json(
    json_path=hparams["valid_json"], replacements={"data_root": data_folder},
)
# We also sort the validation data so it is faster to validate

valid_data = valid_data.filtered_sorted(
    sort_key="duration",
    reverse=True,
    key_max_value={"duration": hparams["avoid_if_longer_than"]},
    key_min_value={"duration": hparams["avoid_if_smaller_than"]},
)

test_data = sb.dataio.dataset.DynamicItemDataset.from_json(
    json_path=hparams["test_json"], replacements={"data_root": data_folder},
)
```

```
# We also sort the validation data so it is faster to validate
test_data = test_data.filtered_sorted(
    sort_key="duration",
    reverse=True,
    key_max_value={"duration": hparams["avoid_if_longer_than"]},
    key_min_value={"duration": hparams["avoid_if_smaller_than"]},
)

datasets = [train_data, valid_data, test_data]

# 2. Define audio pipeline:
@sb.utils.data_pipeline.takes("wav", "start_seg", "end_seg")
@sb.utils.data_pipeline.provides("sig")
def audio_pipeline(wav, start_seg, end_seg):
    info = torchaudio.info(wav)
    start = int(float(start_seg) * info.sample_rate)
    stop = int(float(end_seg) * info.sample_rate)
    speech_segment = {"file" : wav, "start" : start, "end" : stop}
    print(f" start {start}, end {stop}")
    sig = sb.dataio.dataio.read_audio(speech_segment)
    print(sig.shape)
    if info.num_channels > 1:
        sig = torch.mean(sig, dim=1)
    resampled = torchaudio.transforms.Resample(
        info.sample_rate, hparams["sample_rate"],
    )(sig)
    return resampled

sb.dataio.dataset.add_dynamic_item(datasets, audio_pipeline)

# 3. Define text pipeline:
@sb.utils.data_pipeline.takes("wrd")
@sb.utils.data_pipeline.provides(
    "tokens_list", "tokens_bos", "tokens_eos", "tokens"
)
def text_pipeline(wrd):
    tokens_list = tokenizer.sp.encode_as_ids(wrd)
    #tokens_list = tokenizer.encode_as_ids(wrd)
    yield tokens_list
    tokens_bos = torch.LongTensor([hparams["bos_index"]] + (tokens_list))
```

```
            yield tokens_bos
            tokens_eos = torch.LongTensor(tokens_list + [hparams["eos_index"]])
            yield tokens_eos
            tokens = torch.LongTensor(tokens_list)
            yield tokens

        sb.dataio.dataset.add_dynamic_item(datasets, text_pipeline)

        # 4. Set output:
        sb.dataio.dataset.set_output_keys(
            datasets, ["id", "sig", "tokens_bos", "tokens_eos", "tokens"], #tokens_list
        )
        return train_data, valid_data, test_data


    if __name__ == "__main__":

        # Load hyperparameters file with command-line overrides
        hparams_file, run_opts, overrides = sb.parse_arguments(sys.argv[1:])
        with open(hparams_file) as fin:
            hparams = load_hyperpyyaml(fin, overrides)

        # If --distributed_launch then
        # create ddp_group with the right communication protocol
        sb.utils.distributed.ddp_init_group(run_opts)


        # Create experiment directory
        sb.create_experiment_directory(
            experiment_directory=hparams["output_folder"],
            hyperparams_to_save=hparams_file,
            overrides=overrides,
        )



        # Defining tokenizer and loading it
        tokenizer = SentencePiece(
            model_dir=hparams["save_folder"],
            vocab_size=hparams["output_neurons"],
```

```
    annotation_train=hparams["train_json"],
    annotation_read="wrd",
    annotation_format="json",
    model_type=hparams["token_type"],
    character_coverage=hparams["character_coverage"],
)

# Create the datasets objects as well as tokenization and encoding :-D
train_data, valid_data, test_data = dataio_prepare(hparams, tokenizer)

# Trainer initialization
asr_brain = ASR(
    modules=hparams["modules"],
    hparams=hparams,
    run_opts=run_opts,
    checkpointer=hparams["checkpointer"],
)

# Adding objects to trainer.
asr_brain.tokenizer = tokenizer

# Training
asr_brain.fit(
    asr_brain.hparams.epoch_counter,
    train_data,
    valid_data,
    train_loader_kwargs=hparams["dataloader_options"],
    valid_loader_kwargs=hparams["test_dataloader_options"],
)

# Test
asr_brain.hparams.wer_file = hparams["output_folder"] + "/wer_test.txt"
asr_brain.evaluate(
    test_json,
    min_key="WER",
    test_loader_kwargs=hparams["test_jsonloader_options"],
)
```