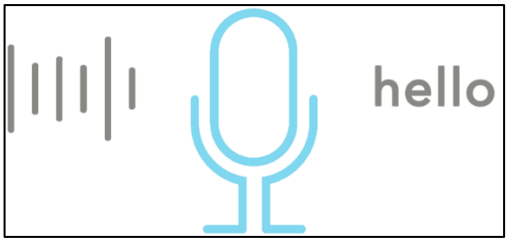
Machine learning models for automatic speech recognition



Jean-Baptiste CHANIER

Université de Paris, Master 2 « Machine Learning pour la science des données »

Aubay, “Aubay Innov’”

12th June 2022

To the attention of Mr. Aghiles

Acknowledgement

I would like to thank Eric REMILLERET, my tutor, and Didier GIOT, for giving me the opportunity to join Aubay Innov', which turned out to be a fascinating and instructive experience. They also ensured that my integration in the company was going well, so that I could work in pleasant conditions.

In addition, for their wise advices, constant availability and the attentive support they were able to put in place, I would like to thank Pascal GE and Florian PLISTA who, as artificial intelligence engineers, were responsible for the projects that I joined during this work-study year.

I pay tribute to Anne-France GALLAND, who made a significant contribution to the monitoring and organisation phase of the projects, through her interventions at the weekly meetings and her managerial skills in general.

I would also like to thank:

- The Aubay general direction for their welcome and their listening
- Ophélie CHEVALIER, campus manager, for all the events she has been able to organize and which have helped to establish a warm environment in the unit

Finally, it seems natural to thank the other Aubay Innov' students with whom I had the opportunity to exchange, for their constant good humor and their availability, and in particular those of my team for their kindness and help.

Table of Contents

Abstract

This document presents the work I did during the in-company part of my year as a work-study student, which took place between September 6, 2021 and August 18, 2022 in an “Entreprise des services du numérique (ESN)” called Aubay, located in Boulogne-Billancourt.

During this year, I worked successively on two in-house projects and I have chosen to focus the content of this report on the algorithms, methods, results and analyses that I have studied / obtained during the second project, to which I was assigned from February 2022.

The team I joined during this period worked on an application that, by using data science and applied mathematics models, would allow blind or visually impaired people to partially recover the ability to identify objects / people in their field of vision, and to move towards close points of interest. The project, called "FYW" for "Find Your Way", was part of Aubay's research and development unit, called "Aubay Innov'".

Personally, I have chosen to focus on tasks related to the study of language elements, with the objective of allowing a simplified interaction with the user. More precisely, my work has consisted in the study and implementation of methods allowing the recognition of words included in an audio sequence, a task regularly referred to as "Speech to text", or "Automatic Speech Recognition" (ASR) in research articles [26] [27] [28].

The first part of my in-company experience was devoted to the study of these methods, which are part of the branch of data science dedicated to the analysis of language components, i.e., Natural Language Processing (NLP).

As a result of the state-of-the-art phase and with the members of my team, we identified two types of possible solutions:

- Training a new model using a well-known ASR framework called Kaldi, which offers concrete functionalities for the whole phase of preprocessing (extraction of features like mel frequency cepstral coefficients (MFCC), or Feature space Maximum Likelihood Linear Regression (FMLLR)), training (choice of models implemented in PyTorch), and prediction (use of decoding graphs called “finite-state transducer”). We have chosen to use two common deep-learning architectures: multilayer perceptron (MLP) and long short-term memory (LSTM).
- Fine-tuning of a large state-of-the-art model available on the Huggingface [32] NLP platform. The chosen model is XLS-R [28], combining convolutional layers with cell transformers (using the attention mechanism). It was trained and shared by Facebook's AI labs in 2021.

We were able to test each of these methods in particular with a French ASR dataset called M-AILABS [38], which allowed us to compare the obtained models, both in terms of pure performance with a metric such as the "Word Error Rate" (WER), and in terms of prediction time during production. Our conclusion is that the re-training of the Huggingface model offers better guarantees, both in terms of the robustness of the predictions and the execution time.

Finally, the objective of the last part of this experience was to ensure the integration of the obtained ASR models with the other algorithms implemented by my colleagues within the complete application developed for the project.

Acronyms

ASR	Automatic Speech Recognition
AST	Automatic Speech Translation
BU	Business Unit
CMLLR	Constrained Maximum Likelihood Linear Regression
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CRDNN	Convolutional Recurrent Deep Neural Network
CTC	Connectionist Temporal Classification
CV	Computer Vision
DCT	Discrete Cosine Transform
DNN	Deep Neural Network
ESN	Entreprise des services du numérique
FFT	Fast Fourier Transformation
FMLLR	Feature space Maximum Likelihood Linear Regression
FST	Finite State Transducer
FYW	Find Your Way
GELU	Gaussian Error Linear Unit
GMM	Gaussian Mixture Model
GPU	Graphics Processing Unit
GRU	Gated Rated Unit
HMM	Hidden Markov Model
IAF	Artificial Intelligence Factory
LPC	Linear Predictive Coding
LPCC	Linear prediction cepstral coefficient
LSTM	Long Short-Term Memory
MFCC	Mel-Frequency Cepstral Coefficients
MLP	Multi-Layer Perceptron
NLG	Natural Language Generation
NLP	Natural Language Processing
PER	Phoneme Error Rate
PoC	Proof of Concept
RELU	Rectified Error Linear Unit
RNN	Recurrent Neural Network
SER	Speech Emotion Recognition
SGD	Stochastic Gradient Descent
SLAM	Simultaneous Localization and Mapping
SVM	Support Vector Machine
TAL	Traitement automatique du langage
TTS	Text To Speech
VQA	Visual Question Answering
WER	Word Error Rate
WMF	What’s My Feeling
XLS-R	Cross-Lingual Speech Representation
YOLO	« You Only Look Once » Neural Network
YOLOR	« You Only Learn One Representation » Neural Network

Glossary

Audio sampling: Process of transforming a continuous musical signal into a discrete digital file

Monophonic: Based on a single channel for transmission. For example, in the case of a musical piece, it would signify that no notes are played simultaneously

Loss: Value of the error function, used to update weights of a deep neural network through gradient descent procedure. The loss function needs to be differentiable

Hyperparameter: Values used to describe the training conditions of the model. They are set before training and do not change during training, so they should be differentiated from the model weights

Learning-Rate: Multiplying coefficient of the gradient of the cost function in the stochastic gradient descent algorithm, influences the update of the model's coefficients

Epoch: Complete (and single) run of the gradient descent algorithm on each of the samples in the training set. Typically, neural network trainings are run on multiple epochs

Batch: Number of training input data to be presented to the model before a new update of its weights

Performance Metric: Measure of the model performance, often chosen to be easily interpretable by humans

Optimizer: Algorithm using the learning rate and other hyperparameter to update values of deep neural network, has the role of avoiding local minima of loss function

Overfitting: Case where the model loses its ability to generalize by learning too deeply the distribution of the data with which it was confronted during its training

Regularization: Method that aims to avoid overfitting by voluntarily reducing the complexity of the model during training, in order to avoid "rote" learning of the data

Drop-out: Regularization method that increases the robustness of model treatments by deactivating some neurons during training

Chunking: Action of extracting short phrases from given sequence of words

Weight decay: Multiplier factor of the sum of the squares of the model weights in the regularization terms in the loss function. A large value leads to a strong penalization, and thus an evolution of the model weights towards 0, meaning a reduction of the model complexity

Momentum: Additional term to the simple calculation of the gradient allowing to take into account the values encountered for the last batches. Allows to direct the gradient descent towards a minimum in a faster and more direct way

Normalization: Reduce a set of values to realizations of a standard normal distribution. In the case of deep learning, can be applied for the values of different samples of a batch or different neurons of a layer and allows the model to handle more easily the extreme values in the gradient calculations.

I – Presentation of the internship

I.1 – Aubay and “Innov” unit

My work-study experience took place in a company called Aubay, for a duration of 11 months (from the 6th of September 2021 to the 18th of August 2022). In details, Aubay is an ESN (“Entreprise des services du numérique” / “Digital Services Company”) that was founded by Christian Aubert in 1998 and whose headquarters are located at the 13 rue Louis Pasteur at Boulogne-Billancourt.

The company is specialized in domains related to finance, insurance and banking, however, it is also involved in diverse markets, such as telecoms, services, network, energy, transports. Specifically, its mission with regards to its clients is to “accompany the transformation and modernization of information system”.

In 2022, the company employs 7306 workers, with 2728 of them located in France. According to its website, Aubay is implanted in 7 European countries, and has realized a turnover of **470,6 M€ in 2021.** In fact, in recent years, the company has enjoyed a “strong rise” with a registered growth of 10,4% in 2021, which coincides with a constant augmentation in the company effectives, evolving from 4600 employees in 2015 to more than 7000 nowadays.

In September, I had the privilege to continue my experience as part of the “Aubay Innov’” unit, which is the Aubay France’s division dedicated to research and innovation. Its members are involved in various projects, each related to data science, data analysis or any other field related to digital new technologies. The official aim of the unit is to ‘acquire the knowledge and know-how for building long-lasting and innovative solutions adapted to your future needs.’.

Each year, this unit gives chances to dozens of trainees to improve or perfect their knowledge relative to data science by letting them the opportunity to discover and experiment the most innovative technologies available in research fields. Moreover, the “Aubay Innov’” unit is widely considered as a source of recruitment for the company, which was willing this year to engage roughly 800 new collaborators in France.

Figure 1: Aubay's logo

Figure 2 : Aubay's headquarters location

I.2 – “What’s my feeling” project

During the first part of my work-study year at Aubay, i.e. the period between September 2021 and February 2022, I joined the Innov' unit project entitled "What's My Feeling (WMF)", the subject of which is to recognise human emotions automatically using machine learning models.

This project, although interesting, seemed to me less rich in terms of content compared to the one I joined later, so I chose not to document this experience too much, in order to focus the substance of this report on the task of the second project I was lucky enough to participate in. Thus, the WMF project will only be mentioned in this report in this brief paragraph.

For this session, the WMF team consisted of a supervisor from the Aubay AI unit, Pascal GE, and 5 interns (myself included). As the previous teams in this project had already explored the issue of recognition from visual information (videos and photos), we chose to focus on an equivalent task, but performed only from audio signals, a task called "Speech Emotion Recognition", or SER.

Thus, during the state-of-the-art phase, we were able to discover the classification with 7 emotions considered as major, namely joy, sadness, fear, surprise, anger, disgust and finally neutral emotion. During this period, I was particularly interested in articles dealing with multilingual procedures, in order to determine whether emotion recognition was similar between different languages. More generally, we learned a great deal about methods for extracting features from audio signals, the main one being the representation of audio signals as mel-spectrograms, which are a class of audio spectra based on the frequency analysis of the audio waveform. This method has thus made it possible to transpose audio signals into images, which has largely influenced the choice of models for this type of task. We were able to train, optimize, and put to the test different types of models, including SVMs and neural networks, such as MLPs or CNNs, and finally a particular architecture, based on the association between convolutional modules and transformer layers.

The results of our models on two test sets from SER datasets, RAVDESS (English language) [1] and CAFE (French language) [2], are illustrated below (Figure 3), the metric here being accuracy:

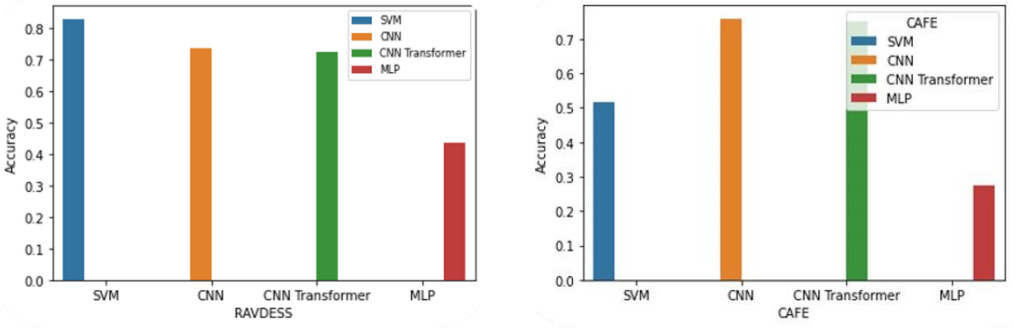


Figure 3 : SER models results

In SER, models obtaining accuracy around 80% are considered to be efficient since it is commonly accepted that humans obtain scores between 85% and 90% of accurate recognition on most datasets encountered in practice.

I.3 – “Find Your Way” project

I.3.a) – Purpose of the project and team

The “Find Your Way” (FYW) project, which represents the experience I will detail in this document, was launched in February 2022 and aims to create an application based on algorithms that can recognise elements in the immediate environment of a visually impaired user and thus orientate and guide them in their movements. The targeted functionalities include first of all the recognition of the number and type of objects (but also people) in the environment close to the user, the estimation of the depth of the scenery but also the conservation in memory of the displacements carried out by the user as well as the proposals of path corresponding to its needs, the whole responding to a system of vocal commands.

During the summer of 2022 campaign, the team for FYW was composed of thirteen people, myself included: Firstly, there was our direct supervisor, Zachary JESSNER, who is an IA engineer at Aubay since 2017. He was present during all our meetings to closely follow our progresses, and represented a great help when it came to issues related to technical considerations. However, for personal reasons, he was later replaced by Florian PLISTA, who came from the same unit of the company, thus presenting even qualifications, who was able to fulfil a similar role, starting from April 2022. The rest of the team were interns, coming from various engineering schools or universities, having all recently studied domains related to machine learning, data management or onboard system, and with each of us in our final year of master.

The team itself was supervised by Eric REMILLERET and Didier GIOT, in charge of the Aubay Innov’ unit as a whole and both long-term Aubay collaborators.

I.3.b) – Organisation of the project

The project in its current configuration (May 2022) is divided into 3 proofs of concept (PoCs):

- Computer Vision (CV): Algorithms for object recognition (especially doors) through neural networks (YOLO [3] / YOLOR [4]), path design, depth estimation and speed recognition. The role of this POC is to manage the immediate environment (field of vision by the cameras) of the user, providing him with a means of integrating information on the objects/people around him or the potential surrounding dangers (obstacles)
- Simultaneous Localization and Mapping (SLAM): Algorithm [5] for identification of points of interest in the environment allowing its complete reconstruction in three dimensions to estimate the path taken by the user, and propose usage of memory maps, that could permit the saving of previously encountered positions. Paths to go back to specific positions are calculated using the A* algorithm.
- Natural Language Processing (NLP): Language processing models enabling interactions with the user through voice recognition and command generation systems

Ultimately, the NLP POC, that I joined at the launching of the project, should contain the following components:

1. Model of "**Automatic Speech Recognition**" (ASR), or "Speech to Text", used to identify and transcribe the words contained in an audio sequence.
2. Model of "**Natural Language Generation"** (NLG), capable of conceiving sentences integrating information. This function would allow the application to intervene with an audible signal even without a request from the user: One could for example think of the case where the user would unknowingly move towards a wall and the system would try to prevent him/her from doing so.
3. Model of "**Visual Question Answering**" (VQA), allows a textual question to be answered using one or more images as a source of information. This model would thus have the task of taking into account the user's requests (example: "How many people are in the room", answer: "8").
4. Model of "**Text to Speech**" (TTS), leading to the enunciation of a textual sequence.

Due to time constraints and limited manpower, we have chosen to focus on specific part of the NLP section. Thus, this document only covers solutions envisaged and tested for the ASR problem (Speech to text), on which our team, composed of two other interns, Jean-Noël CLINK and Miora RASOLOFONERA, with the addition of myself, worked between February and June 2022.

The technical constraints on the characteristics of the audio files, defined at the beginning of the project, are as follows: Sampling at 16 kHz, encoding on 16 bits, monophonic.

Moreover, all the algorithms/models that will appear in the final application of the FYW project must be tuned in French. However, for this work, we have decided to first deal with resources in English, as all the guides of the framework used correspond to this language.

I.4 – Workflow at Aubay Innov'

Internships (or work-study in my case) in the Aubay Innov' unit generally follow similar patterns and are often divided into three characteristic distinct parts:

- State of the art: Given a particular specific task, phase of exploration of available solutions to identify the best of them. It is mostly a time of reading throughout dozens of research articles, understanding concepts that surrounds the task and methods that are currently giving best results.
- Proof of Concept: Following the state-of-art phase, selected methods / algorithms are implemented, tested and optimized. At first, the implementation strictly follows structure of the models proposed by researchers, however, regarding the mandatory adaptation phase to our hardware, datasets and resources available, and our will to add an innovative touch, final kept models were often quite differing from their original inspirations.
- Project Phase: In contrast with the first two parts that are carried out by groups of generally either one or two workers (three in our case), the "project phase" shall concern every member of the team simultaneously and in a common effort, as it represents the gathering and proper combination of every PoCs from the current session.

As a final step, it is supposed to result in the elaboration of a fully functional application which must allow any user to be able to use the methods that we implemented during our internship.

As none of the NLP team members had any experience with ASR pipelines, we chose to extend the state-of-the-art phase until the beginning of August, in order to make sure we understood the ins and outs of the frameworks/libraries/algorithms we would be working with.

Moreover, an important part of our state of the art is dedicated to the study of Visual Question Answering (VQA) articles, i.e. algorithms that allow to bring an answer to a question using images and Natural Language Generation (NLG), i.e, generation of sentences based on the incorporation of specific textual information, that we also thought to implement during the PoC phase.

However, due to tight time constraints, we quickly realized that the best thing to do was only to focus on the implementation of the French ASR models, thus abandoning the testing phases of the VQA and NLG algorithms.

Table 1 shows the planning of the work done for this session on the FYW project:

Table 1 : Official planning working on the FYW project

Month	March	April	May	June	July	August
Phase	State of the Art	State of the Art / Proof of Concept	Proof of Concept	Project Phase	Project Phase	Conclusion of the internship

II – State of Art: Automatic Speech Recognition

For this POC we decided, on the one hand, to implement the Kaldi tool [6] which currently represents the most popular ASR toolkit. Our objective was to get to know this tool and then to implement the Pytorch-Kaldi framework [7], based on the paper written by Mirco RAVANELLI in 2019, in order to use it on a French dataset.

II.1- The Kaldi toolkit

Kaldi is an open-source toolkit made for dealing with speech data. It's being used in a wide range of voice-related applications, but mostly for speech recognition.

The toolkit is constantly updated and further developed by a large community. Kaldi is written mainly in C/C++, but the toolkit is wrapped with Bash and Python scripts.

Kaldi is divided in three parts:

- Preprocessing and Feature Extraction: it consists in identifying the sound of human speech and discarding any unnecessary noise. Kaldi proposes the use of the following features: The content of each audio utterance is represented with **mel-frequency cepstral coefficients** (MFCC) **[8]** and **cepstral mean and variance normalization** (CMVN) **[9]**, while **I-Vectors** **[10]** are used as a means of normalizing samples, taking into account the audio utterance characteristics.

- The Model: Kaldi's model can be divided into two main components. The first part is the **Acoustic Model**, which will transcribe the audio features that we created into some sequence of context-dependent phonemes. The second part is the **Decoding Graph**, which takes the phonemes and turns them into lattices.
- The Training Process: After creating the deep neural network (DNN) acoustic model we can train it to transform the DNN output into the desired lattices.

Figure 4 shows the main components of Kaldi and the steps involved in producing an RSA model with the framework.

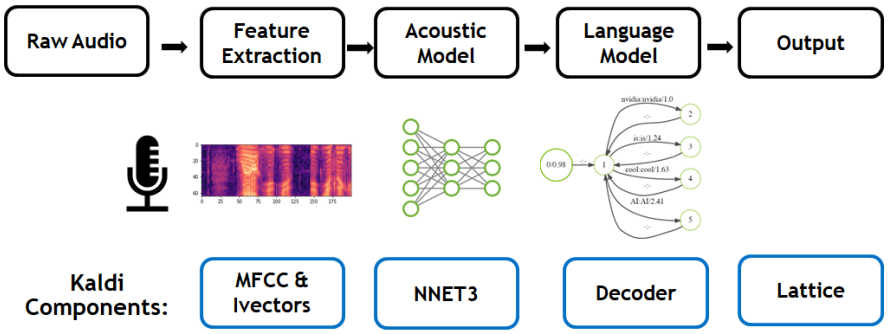


Figure 4 : Options from Kaldi

II.2- Datasets

II.2.a) English ASR Dataset: Librispeech

The "Librispeech" dataset [11] is a frequently encountered resource in ASR. It is a set of .wav files representing together more than 10 000 hours of audio in English language. The audios correspond to our technical constraints since all the files have been sampled at 16kHz.
The data, from the "Librivox" project [12] which makes audio books freely available, were collected by Vassil Panayotov and Daniel Povey in 2015 and presented in the paper "LibriSpeech: an ASR corpus based on public domain audio books" at the ICASSP (International Conference on Acoustics, Speech, and Signal Processing).

The CreativeCommons license of the dataset is CC BY 4.0, which indicates that the data is set to be "remix, transform, and build upon the material for any purpose, even commercially.". The dataset has played a fundamental role in the development of Kaldi, being used in the official tutorial for the framework (and by extension, Pytorch-Kaldi). The preprocessing of this dataset (processing of audio signals, extraction of features) is therefore easily applicable since all the necessary scripts are already available after installation of the git. The use of this dataset was thus unavoidable and we obtained our first results on this one.

II.2.b) French ASR Dataset: M-AILABS

The "M-AILABS" or "M-AILABS Speech Dataset" [36] was collected in 2019 by Imdat Solak. The audio samples come from the "Gutenberg Project" [35] and "Librivox" projects, the former offering the texts of public domain books, and the latter providing the associated audio.

The result of this association is a free dataset, free of charge, and freely accessible to the community interested in machine learning. The books in this dataset were published between 1884 and 1964.

The content of the multilingual dataset represents almost a thousand hours of audio and associated transcripts. Particular attention is paid to the gender (male/female) and the number of speakers from which each audio originates. The audios respect our specifications (16kHz sampling, wav format, monophonic).

In the French section of the dataset, the characters used are those of the ASCII table, with the addition of characters specific to the French language. These are shown in Figure 5:

'àâæçéèêëîïôœùûüÿŸÜÛÙŒÔÏÎËÊÈÉÇÆÂÀ'

Figure 5 : Characters added to M-AILABS

The French section also contains over 190 hours of audio, through the experience of 5 different speakers. It should be noted that the transcriptions do not include transliterations (rewriting of graphemes according to an international alphabet) and that number normalization has not been applied.

II.3- Methodology from Kaldi

II.3.a) Lexicon and phonemizer

In the field of phonetic, a **phoneme** is defined as "the smallest discrete unit that can be isolated by segmentation in the spoken chain". When accurately ordered, they can be used to represent any spoken sequence. It should be noted that phoneme tables differ according to the language they tend to cover: For example, English records 44 distinct phonemes while French has only 36.

In addition, the transcription of a phoneme in writing is referred to as a **grapheme**. In English, these are between 1 and 4 letters long, and make up the written form of words as they appear in texts.

As explained above, the Kaldi framework requires a number of resources before a training of an ASR model can be launched, one of them being a lexicon corresponding to the target language, containing as well the translation into phonemes of each of the graphemes involved. These documents are included in the original Kaldi git for the English Librispeech dataset, so we only had to deal with the M-AILABS processing. For this one, we started by establishing a lexicon of the different tokens present in the transcripts of our dataset.

The complete list contains 53 600 different words. In a second step, we used the "espeak-ng" backend [36] to phonemize the previously created list. It includes a section especially dedicated to the French language. The operation also takes into account slurs, so that the same word can have several different phonemizations. The result is a lexicon, contained in a .txt file with special formatting, which includes 61 438 phoneme transcriptions of the original words.

The Figure 6, taken from the .txt of the lexicon for the M-AILABS dataset, illustrates the phenomenon of multiple possible phonemes representations for the same word:

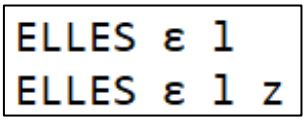


Figure 6 : Ambiguous phonemization

This can be explained by the different pronunciations of the following sentences (without or with slur):

- "Elles descendent …", the word « ELLES » is pronounced without slur, hence the transcription without the third phoneme
- "Elles ont ...", the slur between « ELLES » and « ONT » is taken into account, corresponding to the transcription including the third phoneme

II.3.b) Language Model / N-gram

In natural language processing, a model that is able to model the distribution of sequences of words is referred to as a "language model". The most classic task for language models consists in the prediction of the word that follows an already known words sequence. Language models can come in a variety of forms, some based on statistical considerations, others on models belonging to the category of neural networks. However, regardless of the architecture of the model, the training procedure for language models automatically requires large amounts of textual data.

N-gram models [13] are commonly defined as a "contiguous sequence of n elements of similar type". The index N is referred to as the "order" of the n-gram. They are obtained as a subpart of a given sequence, for example, if we consider the sequence "Je nage tous les jours" and that we want to extract words N-Grams of order 2 (or bigrams), it comes the following list: ["Je nage", "nage tous", "tous les", "les jours"].

In NLP, n-grams are often considered to be the simplest language model: Indeed, the use of these do not involve any training phase. However, once extracted from large corpora of texts, n-grams are regularly used to predict the word that follows a sequence of known words: For a given history of n-1 words, denoted $w_{i-n+1}^{i-1} = w_{i-n+1} \dots w_{i-1}$, the estimated probability of appearance of a designated word is obtained using the maximum likelihood estimation, calculated on the training corpus according to the formula of Equation 1:

$$P\left(w_i \middle| w_{i-n+1}^{i-1}\right) = \frac{N(w_{i-n+1}^{i-1}\, w_i)}{\sum_{w_j \in V} N(w_{i-n+1}^{i-1}\, w_j)}$$

Equation 1: Probability calculation from n-gram

Kaldi proposes a WER rescoring functionality using a correction of the predicted texts by using a language model. In the official tutorial, which deals with the Librispeech dataset, the rescoring uses several n-grams models, trained in English and of order 2, 3 and 4. It should be noted that other types of language models, notably neural, can also be used with the framework.

In addition, the authors of Kaldi have imposed a file type (".arpa"), specially optimised to store and facilitate the use of n-grams language models. Their extraction from our French dataset was made possible by the use of a github created by Mr. Kenneth Heafield [14], who implemented algorithms in C++ allowing extraction from .txt, smoothing (avoiding problems in case of unmet n-gram) of n-grams, and storage in .arpa files.

II.3.c) Extraction of features: MFCCs, I-vectors and FMLLR

The first step in the ASR pipelines from Kaldi is systematically the extraction of features, which is used to extract useful information from the samples and to discard what might disturb the model, in particular background sounds, variations in tone due to emotions, or variations in timbre due to different speakers.

II.3.c.i) MFCCs

One of the fundamental problems for the feature extraction phase is that whenever a human being generates sounds, these are filtered and modified by many phisiological parameters, such as the vocal tract, the tongue but also the teeth, etc. The resulting shape determines the sound that comes out. If we can properly characterize this shape, it should give us an accurate representation of the involved phonemes. The shape of the vocal tract is manifested in the envelope of the short-term power spectrum, and the role of the MFCCs is to accurately represent this envelope.

MFCCs, standing for “Mel-Frequency Cepstral Coefficients” [8] are a widely used feature in most tasks that deals with audio data. They were introduced by Davis and Mermelstein at the beginning of the 1980s, and have remained ever since at the forefront of technology. Prior to the introduction of MFCCs, linear prediction coefficients (LPCs) [15] and cepstral linear prediction coefficients (LPCCs) [16] were the main type of feature for ASR, especially when used with hidden markov models (HMM) [18] classifiers.

Here are the six main steps to extract the MFCCs from a raw audio signal:

1. Frame the signal into short frames (typically 25 ms)
2. Fast Fourier Transform (FFT) are calculated for each frame
3. Mel filterbank are applied to the power spectra, and energy in each filter is summed.

The N (in most cases 26) mel-filterbanks are described as triangular filters based on the previously computed FFTs, but having themselves been decomposed, resulting in N subsets. Those subsets are based of the decomposition of the frequency space into N subsets of the mel-scaled frequencies (i.e. the mel-frequency space). The idea behind mel-scales is that the human ear doesn’t perceive pitch differences as well for higher frequencies. For example, the human ear will perceive the pitch difference between 200 Hz and 300 Hz sounds to be much higher than 18 200 Hz and 18 300 Hz sounds, even though the pitch difference is 100 Hz for both cases. The mel-scale is defined with the formula of Equation 2:

$$M(f) = 1125 \ln(1 + f/700)$$

Equation 2: Definition of mel scale

where f is in Hz and M(f) is in Mels.

4. Logartithm of all the filterbank energies are computed
5. DCT, for Discrete Cosine Transform are extracted of the log filterbank energies

The DCT (here : the 2D-DCT, or DCT-II) is commonly used to compress the energy of the 2D input into : either a small subset of the matricial output (usually the upper-left hand corner of the output), or into a 1D vector. Here, the DCT is used to compress the logarithm of the filterbank energies. The formula for DCT-II is shown in Equation 3:

$$X_k = \sum_{n=0}^{N-1} x_n \cos\left[\frac{\pi}{N}\left(n+\frac{1}{2}\right)k\right] \qquad \text{for } k = 0, \ \ldots \ N-1$$

Equation 3: Formula of DCT-II

With x_n representing the elements of the input matrix and X_k standing for the elements of the (1D) DCT output.

Finally, the desired MFCCs are obtained by conserving the 2-13 first DCT.

II.3.c.ii) I-Vectors

I-Vectors [10], or “Identity-vectors”, are used to better understand the variances inside the speaker domain, by (for example) creating a speaker-dependent representation. The interest of this representation lies in the fact that the distinction between the speaker and the channel variability subspace is erased, as the i-vectors allow projection into a common, low-dimensional space. They thus make it possible to attenuate the phenomenon of "session variability", which designates the problem of variation observed between the various audio samples expressed by the same speaker, this being due, among other things, to phonetic variations.

Moreover, due to their dimension-reduction capacities, I-vectors are typically used when the audio samples have a high duration. For the Kaldi framework, they also have the advantage of fixing the input size of the acoustic model, which allows the use of architectures like MLP or CNN.

II.3.c.iii) FMLLR

“Feature space Maximum Likelihood Linear Regression” (fMLLR) [17] are global features commonly used in ASR and also made available as Kaldi options. In some articles, these features are also referred to as "Constrained Maximum Likelihood Linear Regression" (cMLLR).

The application of fMLLRs reduces the differences between the features of different samples due to the intervention of different speakers and takes the form of matrix multiplications, applied in a speaker-adaptive manner. Mathematically, the fMLLRs are obtained from an affine transformation $x \rightarrow Ax + b$, whit x representing the acoustic feature, A and b being determined coefficients. Using the maximum likelihood process, the values described in Equation 4 are recovered:

$$K = \sum_{t,j,m} \gamma_{j,m}(t)\Sigma_{jm}^{-1}\mu_{jm}x(t)^{+} \qquad G^{(i)} = \sum_{t,j,m} \gamma_{j,m}(t)\left(\frac{1}{\sigma_{j,m}^2(i)}\right)x(t)^{+}x(t)^{+T}$$

Equation 4: Formulas of fMLLRs

With Σ_{jm}^{-1} representing the inverse co-variance matrix, $\gamma_{j,m}$ sought coefficients and with $\mu_{j,m}$, $\sigma_{j,m}^{2}$ denoting respectively mean and standard deviation operators.

FMLLRs are particularly effective when used in preparation of hybrid models between a neural network (acoustic model) and an HMM (see decoding graphs).

FMLLRs, frequently compared with other ASR features such as FBANKS or MFCCs, seem to lead to a higher quality preprocessing: Here, a comparison has been implemented, with, in the columns, the preprocessing modes, and in the rows the different model architectures. The results are shown in Figure 12. The dataset used is TIMIT [19] and the metric is the "phoneme error rate" or PER, which is an equivalent of WER for comparison of phonemes sequences. It is clear that the use of fMLLRs leads to a non-negligible improvement in the performance of the models (of the order of 2%).

Table 2 : Contribution of fMLLR to preprocessing

Models/Features	MFCC	FBANK	fMLLR
MLP	18.2	18.7	**16.7**
RNN	17.7	17.2	**15.9**
LSTM	15.1	14.3	**14.5**
GRU	16.0	15.2	**14.9**
Li-GRU	15.3	14.9	**14.2**

II.3.d) Models

We have chosen to use two common deep learning architectures (branch of machine learning dedicated to the study of neural networks), namely the "multilayer perceptron" (MLP) [20], which has a simple architecture, and the "long-term memory" (LSTM) [21], which have the particularity of being recurrent models, able to identify the links between different elements of a sequence.

The structure of a neural network is organized as a succession of layers of neurons, each of which refers to "a set of neurons that do not share any connection with each other." The first layer, called the “input layer” has for sole role the introduction of the input data (a numerical value per neuron, or node, of the input layer) in the architecture of the network and is often not included in the structure because it does not induce any transformation of the signals encountered. The last layer of the network, on the other hand, serves only to present the signal resulting from the modifications of the network, applying a special activation function to it as appropriate. The part of the network responsible for combining and modifying the input signals is therefore only the set of successive hidden layers.

Intrinsically, neural networks can only process information presented as numeric values, which often induces a transformation of the data before it is fed to the network. For example, an image will be introduced into the network in the form of each of the pixel values that compose it.
The most common case of neural networks encountered in practice is the multi-layered perceptron, or MLP. This is also often referred to by the term "Feedforward Neural Network" because each of the neurons in a layer takes as input the signals from all the neurons of the previous layer but sends its output signal to all the neurons of the next layer.

In addition, networks in this category do not have recurrence cells, that is, the connections between nodes in the network do not cycle. Thus, in the network, information only flows in one direction, in this case forward.

The figure 7 presents an example of MLP architecture:

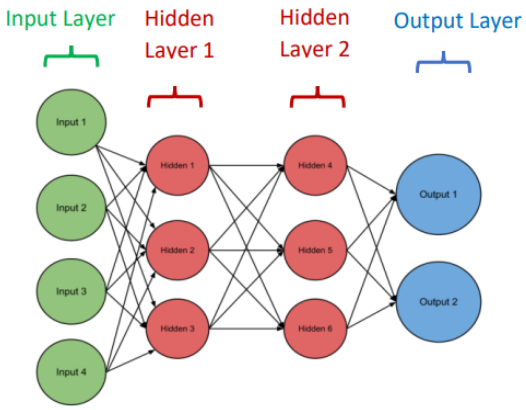


Figure 7 : Three-Layers MLP

Usually, the Multi-Layer Perceptrons / Feedforward Neural Network presented above are not suitable to process texts because their structure for the input layer of the model only allows the processing of sequences of words of fixed size. However, this statement is not verified with the Kaldi framework due to the usage of proper preprocessing (fMLLR, MFCC) as inputs and FST [22] decoding as outputs, which overcome this problem by allowing the size of the network inputs and outputs to be fixed.

The second type of model, called LSTM for "Long-Short Term Memory" [21], was introduced in 1997. It is built upon the recurrence concepts that are used for RNNs, but are optimized to avoid gradient problems (vanishing or exploding) that prevent the model from capturing the subtlety of long-term recurrences (for example: semantic links between words that are in distant positions in a text).

However, the LSTM's architectures present several differences with RNN's: the hidden layer is replaced by a cell, called "Memory cell", denoted as C, which, through a system of successive "gates", has the capacity to discern the information which must be kept for the rest of the training. These "gates", three in number, play the following roles:

• « Forget gate »: Reviews information from h_{t-1} and x_t, and allow the model to choose the information from the previous cell C_{t-1} which is not worth keeping for the rest of the training
• « Input gate »: Discerns which information from the previous cell C_{t-1}, must be updated and proposes candidates for correction, $\tilde{C}_t$
• « Output gate »: Using the results of the two previous gates, process the values from C_{t-1}, by eliminating one part and modifying another. The result of these calculations, noted Ct stands for the current cell outputs

The figure 8, taken from https://www.altoros.com/blog/text-prediction-with-tensorflow-and-long-short-term-memory-in-six-steps/, illustrates the internal computations of a LSTM cell:

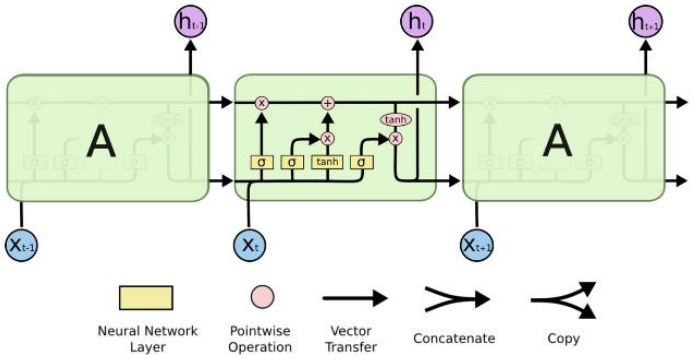


Figure 8 : Memory cell of a LSTM model

II.3.e) Decoding Graphs and Finite State Transducer

Kaldi's model can be divided into 2 main components, the first part is the acoustic model, which is a deep neural network in our case. Its role is to predict a context-sensitive phoneme sequence from the features extracted from the audio signal during the preprocessing phase. The decoding graph is the second main component, it takes the phonemes and turns them into lattices (particular representation of a sequence of words that corresponds to a part of the audio). During this operation, the distribution and probability on the word sequences (calculated from the n-grams) are taken into account, as well as the grammar extracted from the data.

The decoding graph is essentially a Weight Finite State Transducer (WFST) [23]. A finite-state transducer can be defined as a finite automaton with symbols as input and output labels for each of its transitions. In the toolkit, those automates are created with OpenFST [34].

The FST [22] starts in a designated start state and transitions to different states depending on the input, while producing an output according to its transition table. Finite state transducers can be weighted, with each transition labelled with a weight in addition to the input and output labels.

For Kaldi FSTs, context-dependent states are represented by the transition input symbols.In Kaldi dialect, these symbols are called "pdf-ids", and they are represented by numbers. The specificity of kaldi concerns the way they treat the operation that is supposed to guarantee that the FST is stochastic. The stochasticity property refers to the fact that for each state of the FST, the weights sum to one, as in a normalized hidden Markov model. The Kaldi's approach is to avoid weight-pushing but to ensure that each step of the graph creation "preserves stochasticity" in an appropriate sense.

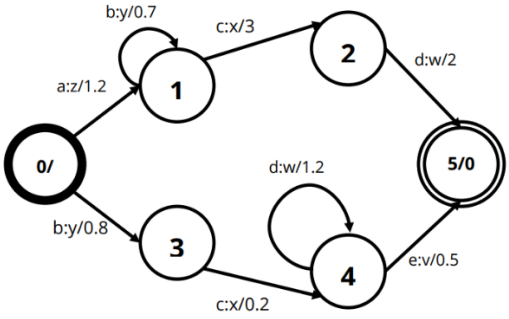


Figure 9 : Representation of WFST with 6 states and 8 transitions

On figure 9, the WFST takes the symbol sequences ab...bcd, and bcd...de as input and produces the symbol sequences zy..yxw and yxdde for the two respective inputs. The transition from state 0 to state 1 consumes the symbol ‘a’ of the input sequence and produces the symbol 'z' of the output sequence with a transition weight of 1.2. It is noted (0,1,a,z,1.2). The transitions from state 1 to 2 and 2 to 5 are written in the same way with the corresponding input and output labels.

II.4- PyTorch-Kaldi

The PyTorch-Kaldi project [7] aims at creating a framework exploiting both the preprocessing and decoding functionalities of Kaldi, as well as the deep neural networks trained with the PyTorch library. More precisely, as the deep neural network part is managed by PyTorch, the feature extraction, label computation, and decoding are performed with the Kaldi toolkit.

PyTorch is defined as an emerging python package that implements efficient GPU-based tensor computations and facilitates the design of neural architectures, thanks to proper routines for automatic gradient computation.

The toolkit purpose is to simplify the process of integration of state-of-the-art acoustic models into Kaldi ASR pipelines. In practice, users can embed their deep learning model and conduct ASR experiment. PyTorch-Kaldi implements hybrid DNN-HMM speech recognizers. The architecture of the framework is illustrated in the figure 10:

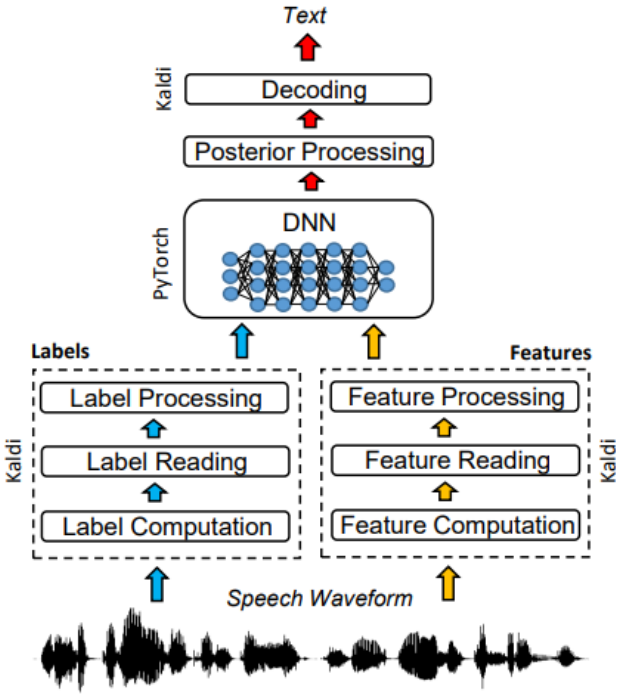


Figure 10 : Pytorch-Kaldi architecture

The feature extraction is performed with Kaldi. The computed coefficients are stored in binary archives and are later imported into the python environment using the kaldi-io. The load-chunk function can then be used to apply transformations to the features, including normalization with respect to the mean and variance, but also the implementation of procedures useful for training, such as the shuffling of samples or the composition of context windows.

The training does not take place directly with the audio samples and their transcriptions, but rather after the application of a forced alignment mechanism between the features extracted by Kaldi (MFCC, fMLRR, i-Vectors, ...) and a contextualised version of the labels, obtained using a phonetic decision tree.

With Pytorch-Kaldi, the dataset is automatically split into chunks, which is a concept close to the one of batches, associating features and transcriptions from different audios, by sampling effect. In this case, an epoch corresponds to the review of a complete set of chunks (which are referred to as "minibatches" in the official PyTorch-Kaldi documentation), playing a role in the updating of the model weights by the mechanism of gradient descent. For each minibatch, the trained neural model will produce during training a vector with probabilities corresponding to the phoneme table of the target language and taking into account the context of the sequence.

Some neural models are natively implemented within the toolkit: Indeed, the current version includes basic architectures such as MLPs but also convolutional models with CNNs or even sequence models with the precursor RNNs and models derived from it, GRUs and LSTMs. For this POC, we decided to test only the MLP and LSTM models.

For the decoding part, the acoustic posterior probabilities generated by the neural network are normalized and then passed to the decoder of Kaldi. They used Word-Error-Rate (WER) as scoring method and is computed with the NIST SCTK [38] scoring toolkit, which is a collection of software tools designed to score benchmark test evaluations of ASR Systems.

II.5- ASR Metric: Word Error Rate

In ASR, as for most branches of NLP whose data are textual, the reference metric is the "Word Error Rate", or WER. This quantifies the actual differences between two texts (in our case, the ground-truth transcription of the audio and the model prediction) and is based on the number of word-level substitutions, deletions, and additions to go from the first text to the second. The formula of WER is presented in Figure 11:

$$WER = \frac{S_W + I_W + D_W}{N_W}$$

Figure 11 : WER formula

With S_W, I_W and D_W denoting respectively the number of substitutions, insertions and deletions of the words involved and N_W the number of words in the original text.

II.6- NLP Library: HuggingFace

II.6.a) Wav2Vec2

Wav2Vec2.0 [26] is a state-of-the-art ASR model that was published in 2020 by Baevski- et al.

The observation that labelled data is sorely lacking in ASR led the authors to design a model capable of being trained in a self-supervised manner. This echoes the way humans learn, who

are able to improve their language skills simply by being in regular contact with native speakers of that language.

The training of the model is separated into two phases:

- Self-supervised training: Unlabelled data is used to improve the audio recognition capabilities of the model. During this phase, the model learns to create speech representations.
- Supervised Fine-Tuning: The pre-trained model is refined on a supervised dataset. This phase allows the model to focus on the recognition/prediction of particular phoneme sequences (here words).

The authors of the paper trained the model on a huge dataset called LibriVox, then fine-tuned their model on different sections of Librispeech, obtaining particularly interesting results:

- 10 minutes of speech: 4.8% of WER in test data
- 100 hours of speech: 2% of WER in test data
- 960 hours of speech: 1.8% of WER in test data

The quality of the pre-training is such that simple finetuning with 10 minutes of audio is sufficient to achieve a WER that would have been state of the art on this dataset in 2018.
The architecture of the model is shown in the following image:

Figure 12 shows the architecture used in the Wav2vec2.0 model:

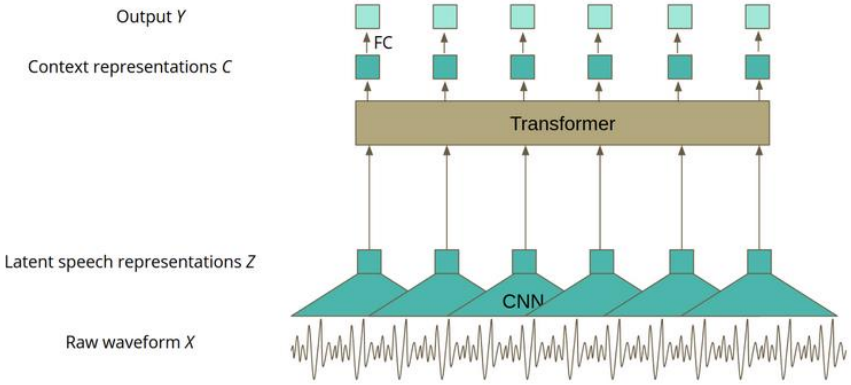


Figure 12: Wav2Vec2.0 architecture

The first layers are convolutional and allow the extraction of latent representations from the audios directly. In a second step, transformer [25] units, able to handle sequenced data, enrich the representations by adding context. The last layer, linear, present only in the final model (not in the unsupervised training), is useful for prediction.

Inspired from the training of the BERT [24] model, the authors used the method of masking parts of the representations during training to make the model more robust. This is done before the CNN representations are given as input to the transformers: A proportion p=0.065 of the timestamps is selected, acting as initial indexes for the masking, which is effective on the next m (10 in the original paper) values of the representation vector.

More generally, this method belongs to the class of "contrastive learning", the aim being to train the model to recognise whether two transformed representations come from the same source or not. The first transformation is carried out by transformers (addition of the context), the second by quantisation (passage from a continuous space to a concrete space).

Quantization comes into play when choosing a representation for a sound sequence: the number of possible sounds is infinite, but the phoneme table in French is finite.

During the first phase of the model training, the optimised loss has two terms, and corresponds to Equation 5:

$$L = L_m + \alpha L_d$$

Equation 5: Loss of Wav2Vec2.0

The first term, L_m, represents the contrastive loss and is shown in Equation 6:

$$L_m = -log \frac{exp(sim(c_t, q_t)/\kappa)}{\sum_{\tilde{q} \epsilon Q_t} exp(sim(c_t, \tilde{q})/\kappa)}$$

Equation 6: Contrastive Loss

Here the formula is very similar to the one used for softmax. K is a fitting constant that is fixed throughout the training. The notation "sim" is assigned to the cosine similarity function. Finally, the notations c_t and q_t stands for respectively the contextual representations (from the transform layers) and the quantised representations. In a way, this formula allows us to assign a probability to the fact that two representations come from the same source, which leads to a direct improvement in the model's performance.

The representations for each phoneme are stored in different "collections" called "codebooks" and are referred to by the authors of the article as "codewords". The formula presented in Equation 7 is used to select the appropriate codewords from G codebooks for a given phoneme sequence:

$$p_{g,v} = \frac{\exp(l_{g,v} + n_v)/\tau}{\sum_{k=1}^{V} \exp(l_{g,k} + n_k)/\tau}$$

Equation 7: Selection of codewords

Here with $l_{g,v}$ representing the entry number v of the codebook g, and n being constants related to realisations of uniform laws.

Thus, the second term, denoted L_d, is the diversity loss and is described in Equation 8:

$$L_d = \frac{1}{GV} * (-H(\bar{p}_g)) = \frac{1}{GV} \sum_{g=1}^{G} \sum_{v=1}^{V} \bar{p}_{g,v} log(\bar{p}_{g,v})$$

Equation 8: Diversity Loss

This operation is equivalent to regularisation, by pushing the model to use as many different codewords as possible. Indeed, the term noted here H, represents the entropy related to the different codewords, the term being minimised when the probabilities $\bar{p}_{g,v}$ are uniforms. Moreover, G and V are respectively the number of codebooks and codewords per codebooks for the original model (2 and 320).

For the pre-training of the model, the authors gathered a huge corpus of unlabelled data: they made a collection between the Librispeech dataset (960 hours of audio) and LibriVox (52 300 hours of audio), which ensures a certain quality for the pre-training phase (unsupervised) of

the model. The authors state that the fine-tuning phase is more incidental, and that the pre-trained model can be made operational with small labelled datasets. In the original paper, they used the 960 hours of audio from the Timit dataset with the Adam optimiser [35].

In addition, the authors involved rescoring using language models, one based on 4-grams, and the other having a neural structure.

The results, as specified above, allow us to state that this model is state of the art in ASR. However, it should be noted that the original version of the model is only available in English.

II.6.b) XLSR

The XLSR [27] model is an improved version of Wav2vec2.0, released the same year (2020) and published by the same laboratory (Conneau et al., Facebook AI). The architecture of the model is very similar, except that XLSR is designed to take into account multilingual corpora by imposing a sharing of discrete tokens between languages.

The authors have made some minor changes in the composition of the model, including the addition of the use of the L2 norm to regularise the representations from the convolutional layers and the use of connectionist temporal classification (CTC), which is a neural architecture tool specifically designed for tackle variable timing in sequence problems, to perform the model prediction.

The model was pre-trained on a collection of multiple datasets, including "CommonVoice" (2000 hours of speech covering 38 languages, including 350 hours in French), "BABEL" (a specialised dataset for African and Asian languages) [30] and "Multilingual Librispeech" (8 languages including French). The French section of this unified dataset represents 354 hours, with the addition of an extra hour for fine-tuning.

Similar to the Wav2Vec2.0 drive, the model is available in several versions, whose architectural components are similar, the versions being differentiated only by the number of weights (number of layers / size) in the model.

The authors chose to compare monolingual XLSR models with the multilingual version. To obtain performance indicators, a new section of CommonVoice was used. The metric used for this procedure is the PER (Phonem Error Rate), which is similar in principle to the WER, but which, rather than two sequences of words, gives an indication of the differences between two sequences of phonemes.

Figure 13 presents the results obtained with several configurations involving XLSR models :

Model	D	#pt	#ft	es	fr	it	ky	nl	ru	sv	tr	tt	zh	Avg
Number of pretraining hours per language				168h	353h	90h	17h	29h	55h	3h	11h	17h	50h	793h
Number of fine-tuning hours per language				1h	1h	1h	1h	1h	1h	1h	1h	1h	1h	10h

Our monolingual models														
XLSR-English	CV_{en}	1	1	13.7	20.0	19.1	13.2	19.4	18.6	21.1	15.5	11.5	27.1	17.9
XLSR-Monolingual	CV_{mo}	1	1	6.8	10.4	10.9	29.6	37.4	11.6	63.6	44.0	21.4	31.4	26.7
Our multilingual models														
XLSR-10 (unbalanced)	CV_{all}	10	1	9.7	13.6	15.2	11.1	18.1	13.7	21.4	14.2	9.7	25.8	15.3
XLSR-10	CV_{all}	10	1	9.4	14.2	14.1	8.4	16.1	11.0	20.7	11.2	7.6	24.0	13.6
XLSR-10 (separate vocab)	CV_{all}	10	10	10.0	13.8	14.0	8.8	16.5	11.6	21.4	12.0	8.7	24.5	14.1
XLSR-10 (shared vocab)	CV_{all}	10	10	9.4	13.4	13.8	8.6	16.3	11.2	21.0	11.7	8.3	24.5	13.8
Our multilingual models (Large)														
XLSR-10	CV_{all}	10	1	7.9	12.6	11.7	7.0	14.0	9.3	20.6	9.7	7.2	22.8	12.3
XLSR-10 (separate vocab)	CV_{all}	10	10	8.1	12.1	11.9	7.1	13.9	9.8	21.0	10.4	7.6	22.3	12.4
XLSR-10 (shared vocab)	CV_{all}	10	10	7.7	12.2	11.6	7.0	13.8	9.3	20.8	10.1	7.3	22.3	12.2
Our Large XLSR-53 model pretrained on 56k hours														
XLSR-53	D_{53}	53	1	**2.9**	**5.0**	**5.7**	**6.1**	**5.8**	**8.1**	**12.2**	**7.1**	**5.1**	**18.3**	**7.6**

Figure 13 : XLSR comparison with ASR models

These values show a clear difference in performance between the monolingual and multilingual models: With the example of French, the monolingual version obtains a PER of 10.4 while the multilingual model results in a performance of 5.0.

II.6.c) XLS-R

In 2021, a researcher team, Wang et al. (from Meta AI, Google AI, Outreach, Huggingface institutes), published the article "XLS-R: self-supervised cross-lingual speech representation learning at scale" [28] in which a new ASR model is presented, again based on the Wav2Vec2.0 structure and called XLS-R. The architecture of the model is represented in Figure14

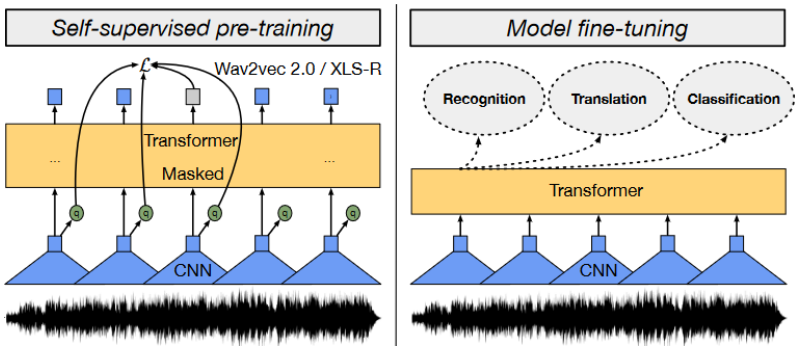


Figure 14: XLS-R architecture, similar to Wav2Vec2.0's

Although the components of the model are similar to those of Wav2Vec2.0 and XLSR, the size of the model has been greatly increased: Thus, if XLSR already had more than 300M parameters, XLS-R, in its largest version, has more than 2B (compared to 300M for its smallest version). Moreover, the authors have experimented with fine-tuning the pre-trained model on other tasks than ASR, among which: Automatic speech translation / AST (Translation of audio samples) or speech classification (Classification of audio samples with regards to the language or the speaker).

For the ASR, the datasets used are the following:

- VoxLingua : 6 600 hours over 107 languages
- BABEL : 1 000 over 17 languages
- Multilingual Librispeech : 50 000 over 8 languages
- CommonVoice : 7 000 hours over 60 languages
- VoxPopuli : 372 000 hours over 23 languages

In total, the complete dataset counts more than 436 000 hours of audio (including 23 973 in French).

In a similar way to XLSR, the predictions of the models dedicated to ASR are operated with a CTC layer.

Evaluated on the 5 ASR datasets separately, and with regards to the WER metric, the performances of XLS-R place it as being state of the art. In the Figure 15, the models are compared to those of the previous articles with the CommonVoice dataset test set:

	es	fr	it	ky	nl	ru	sv	tr	tt	zh-HK	Avg
Labeled data	1h	1h	1h	1h	1h	1h	1h	1h	1h	1h	
Previous work											
m-CPC	38.0	47.1	40.5	41.2	42.5	43.7	47.5	47.3	42.0	55.0	44.5
Fer et al. (2017)	36.6	48.3	39.0	38.7	47.9	45.2	52.6	43.4	42.5	54.3	44.9
XLSR-10	7.9	12.6	11.7	7.0	14.0	9.3	20.6	9.7	7.2	22.8	12.3
XLSR-53	2.9	5.0	5.7	6.1	5.8	8.1	12.2	7.1	5.1	18.3	7.6
This work											
XLS-R (0.3B)	3.1	5.4	4.9	5.1	5.8	6.0	7.2	6.0	4.1	17.0	6.5
XLS-R (1B)	**2.0**	**3.9**	**3.5**	4.1	**4.2**	4.1	5.5	4.4	3.4	15.7	5.1
XLS-R (2B)	2.2	4.0	**3.5**	**4.0**	4.7	**3.7**	**5.0**	**4.0**	**2.9**	**14.8**	**4.9**

Figure 15: Comparison between XLS-R models and previous ASR works

Having been made public in 2021, the original ASR-dedicated XLS-R model, and in particular the French version, is available on the Huggingface platform [32]. In addition, many other models, fine-tuned on other French datasets, are also available online on this platform.

II.6.c) CRDNN

In 2021, on the HuggingFace platform, the NLP research group called Speechbrain [33] (Cambridge, Computer Science Laboratory of Avignon, University of Sherbrooke, Bio-ASP) has posted an ASR model with a particular architecture: CRDNN.

The model is composed of N convolutional blocks (with normalization and pooling) followed by a bi-directional LSTM, providing the attention component, and then a DNN that transforms the representations to a CTC, which provides the final prediction of the model.

On the French section of the Common Voice dataset, the model obtains a WER of 17.70%.

II.6.d) Fine-Tuning

In the “Oxford Languages” dictionary, the term “fine-tuning” is defined as follows: “make small adjustments to (something) in order to achieve the best or a desired performance.”.

Echoing this definition, in deep learning (the branch of data science dedicated to the study of neural networks) however, the term "fine-tuning" refers to the fact of subjecting a second training to a network in order to use the benefits of its first training on a similar task.

This procedure is particularly well suited to the use of transformers as they are often extremely heavy networks, trained with significant resources (several GPUs used for weeks).

It is then possible, often with the help of a slight modification of the network architecture (addition of a fully connected layer as the last layer of the network to modify the model outputs for example), but above all by re-training the model, to orient it towards a task different from those targeted by its first training.

For example, during a previous internship, I had to fine-tune a pre-trained French language model, CamemBERT [31], to use it for erroneous token word in texts recognised by an ICR (Intelligent Character Recognition) model, which is used to perform translation of manually entered text characters into machine-readable characters.

It should also be noted that fine-tuning procedures generally involve a reduced number of epochs, since training too deeply would lead to a loss of information from the first training. For the same reason, it is common to freeze (fix the value of) certain weights, or even entire layers, from the first training in order to ensure that the benefits of the first training are retained.

For this project, we chose to fine-tune the XLS-R model, without modifying the model architecture, because the initial training also involved an ASR task.

III – Tests and results of ASR models

III.1) Kaldi/PyTorch-Kaldi

This part aims to highlight the results we obtained with the MLP and LSTM models, using the LibriSpeech and MAILABS speech datasets, via the PyTorch-Kaldi framework. Both the WERs and the total runtimes will be showcased, for the testing and/or production phases.

As we only managed to fully exploit the features of PyTorch-Kaldi with a custom French dataset (MAILABS) quite recently, we didn't have time to test the MLP and LSTM models with more than 15-20 epochs. The results are still very satisfying!
Also, since the authors of PyTorch-Kaldi didn't provide (reliable) documentation to test the production mode for other models than MLP, we didn't test the production mode for the LSTM model.

III.1.a) LibriSpeech

At first, we tested the PyTorch-Kaldi framework on the LibriSpeech dataset. Indeed, the authors of PyTorch-Kaldi only provided documentation to test their framework on 2 datasets: TIMIT and LibriSpeech. And since the TIMIT dataset isn't free (> 100 €), our only choice was to test PyTorch-Kaldi on LibriSpeech.

This dataset was mostly used as an entry point to test PyTorch-Kaldi, but it can also be used for its associated production mode (in PyTorch-Kaldi) if, for some reason, we need to decode live English audio.

III.1.a.i) Multi-Layered Perceptron

Here are the graphs (Figure 16 and 17) associated with the accuracy and the loss (for 5 epochs or less) with val_0 denoting the validation dataset:

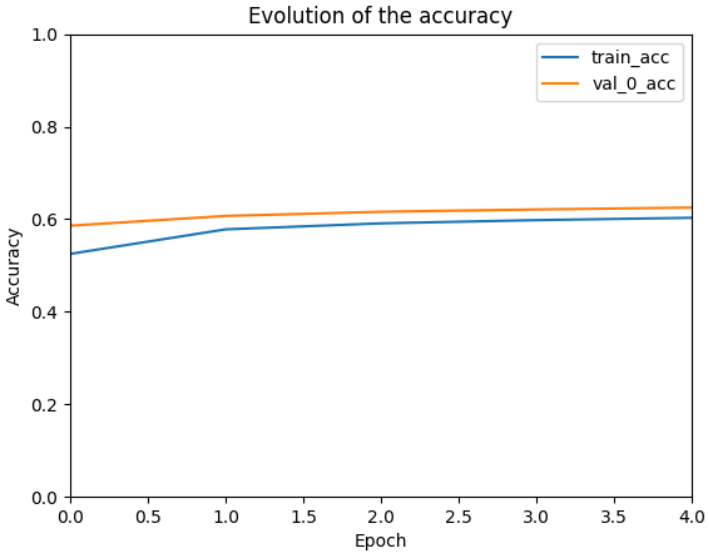


Figure 16: Accuracy for the MLP model on LibriSpeech (5 epochs)

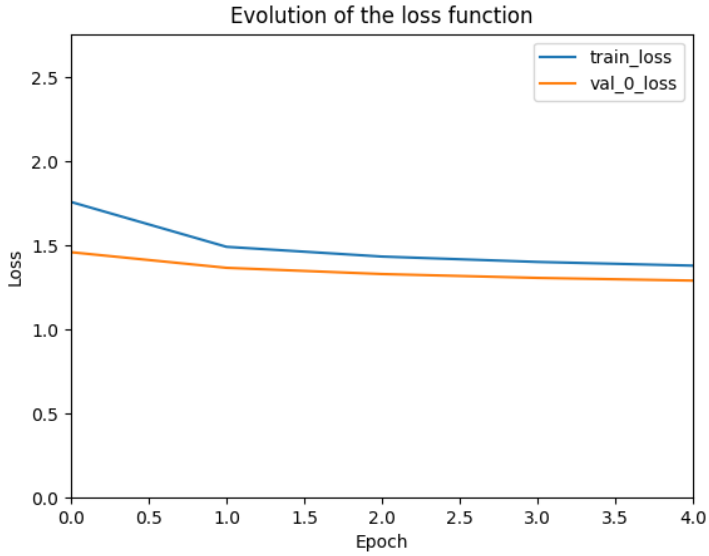


Figure 17: Loss for the MLP model on LibriSpeech (5 epochs)

Table 3 depicts the results we obtain for 1 and 5 epochs (we didn't include the production mode results here since we're not interested in decoding live English audio.

Table 3 : Results of Kaldi MLP models on Librispeech

Epochs	**1**	**5**
Loss	1.760	1.381
Error	0.475	0.397
Validation loss	1.461	1.292
Validation error	0.414	0.375
Training runtime (min)	22	22
WER (%)	**11.33**	**10.25**

Globally, the losses, errors and WERs decrease with the number of epochs (which is comforting). Therefore, here it would be interesting to see what performances we get with more epochs.

III.1.a.ii) Long Short-Term Memory

Here are the graphs (Figures 18 and 19) associated with the accuracy and the loss (for 3 epochs or less) with val_0 denoting the validation dataset:

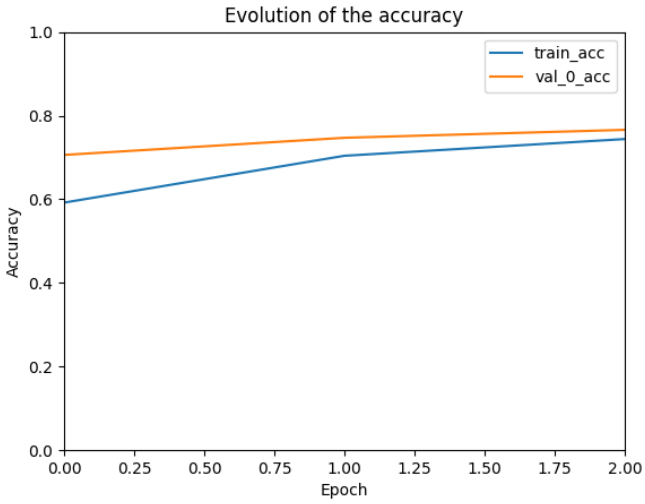


Figure 18: Accuracy for the LSTM model on LibriSpeech (3 epochs)

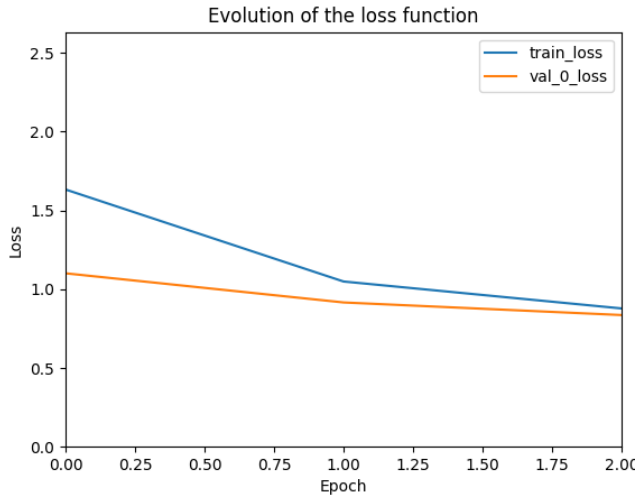


Figure 19: Loss for the LSTM model on LibriSpeech (3 epochs)

Table 4 reports the results we obtain for 1 and 3 epochs:

Table 4: Results of Kaldi LSTM models on Librispeech

Epochs	**1**	**3**
Loss	1.633	0.878
Error	0.408	0.256
Validation loss	1.101	0.836
Validation error	0.294	0.234
WER (%)	**10.33**	**9.97**

Like for the MLP model, the losses, errors and WERs globally decrease with the number of epochs. Therefore, here it would be interesting to see what performances we get with more epochs.

III.1.b) M-AILABS

The end goal is to optimize the production phase with this dataset, i.e. we want to get the lowest WERs, but with also the lowest runtime. Using this French speech dataset allows us to decode live (French) audio in a reasonable amount of time.

For this dataset, we kept approximately the same amount of audio samples for the training, validation (dev) and testing datasets as LibriSpeech, i.e. (respectively): 30000, 3000 and 3000 samples. We did this in order to be able to have a legitimate comparison between both datasets.

III.1.b.i) Multi-Layered Perceptron

The technical characteristics of the model are as follows:

- Number of layers: 5
- Type of layers: Dense
- Number of neurons per layer: 1024
- Training batch size : 128
- Drop-out: 0.15
- Activation function: RELU
- Learning-rate: 0.08
- Optimizer: Stochastic Gradient Descent
- Batch normalization: True
- Layer normalization: False
- Use of momentum: False

The model, although having a simple structure (succession of dense layers) has a heavy architecture, mainly because of the important number of neurons per layer. Such an architecture is required to match the difficulty of the ASR problem.

Here, having only a short time available, we have chosen to impose a high learning-rate, which has the effect of forcing a convergence of the loss in a few epochs: The evolution of the validation loss is almost stationary from epoch 5 onwards, going from a value of 1.753 to 1.689.

The use of the GPU makes it possible to set the batch size at a high value without fearing run out memory while increasing the training speed.

The use of the RELU activation function has become almost systematic in the current deep-learning community, for the advantages offered over a function such as sigmoid, which forces significant quantification of values within the network.

The drop-out, which here takes a relatively low value (0.15 meaning that for each batch, 15% of the neurons of the network are deactivated) allows to avoid the phenomenon of overlearning: The validation loss does not go back to any moment of the training.

Finally, we have chosen to keep the optimizer proposed by PyTorch-Kaldi, i.e. the SGD, which offers performances almost similar to the best optimizers currently available such as Adam or AdamW.

Here are the graphs (Figure 20 and 21) associated with the accuracy and the loss (for 12 epochs or less) with val_0 denoting the validation dataset:

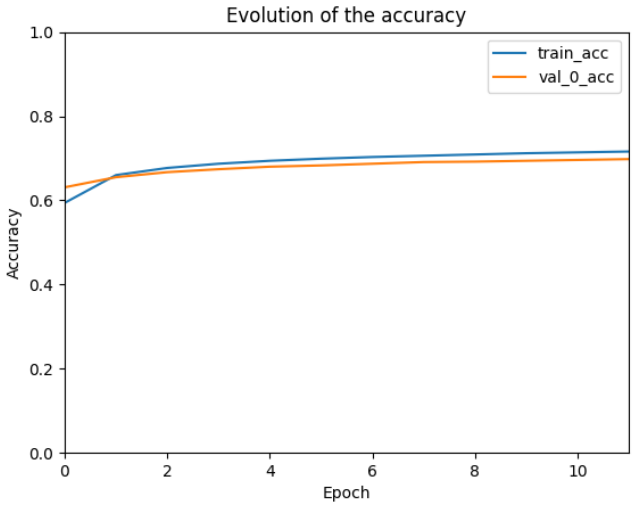


Figure 20: Accuracy for the MLP model on MAILABS (12 epochs)

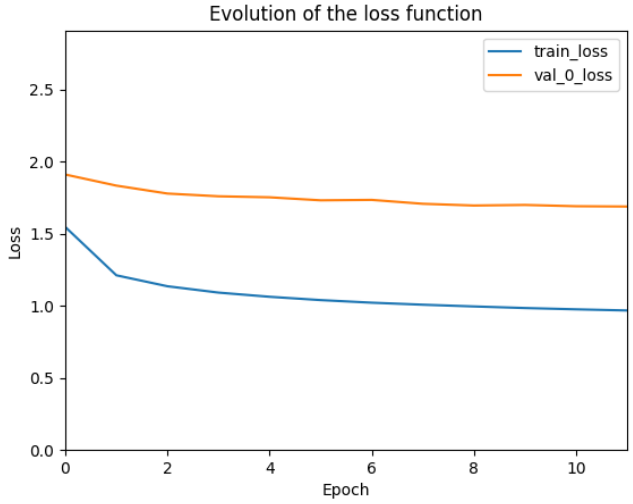


Figure 21: Loss for the MLP model on MAILABS (12 epochs)

The results we get, for 1, 5 and 12 epochs are reported in the table 5:

Table 5: Results of Kaldi MLP models on MAILABS

Epochs	**1**	**5**	**12**
Loss	1.547	1.063	0.968
Error	0.406	0.306	0.284
Validation loss	1.911	1.753	1.689
Validation error	0.369	0.320	0.302
Training runtime (min)	15	15	15
WER (%)	**21.07**	**19.08**	**18.48**
Production runtime (s)	**34**	**50**	**50**
Production WER (%)	37.50	50.00	37.50

As we can see from the previous table, we get better and better losses, errors and WERs as the number of epochs increases. It's also comforting that the WER is proportional to the loss.

We can also notice that, as the number of epochs increases, the production runtime also increases, which is a phenomenon due to the progressive enrichment of the decoding graph. There is thus, through the increase in the decoding time of the model outputs, a trade-off between the WER and the production time.

While the WER performances aren't as good as LibriSpeech (9-10% for LibriSpeech vs 18-21% for MAILABS), they can still be considered as satisfying according to ASR standards.

Moreover, the proposed production times seem to be far too long for the desired use: Indeed, the objective of this PoC is to get as close as possible to real time, allowing to bring a fast transcription of the voice commands of the user of the connected object. Thus, the solution proposed by Kaldi does not seem adapted to our project, which would direct us towards the option of fine-tuning the Huggingface models.

III.1.a.ii) Long Short-Term Memory

Since, for this dataset, we're only interested in the production mode, and because the authors didn't provide documentation to test the production mode for the LSTM model, we are only able to compare the WERs with those obtained using the MLP architecture.
The technical characteristics of the model are as follows:

- Number of layers: 4
- Type of layers: LSTM
- Training batch-size: 16
- Number of neurons per layer: 550
- Drop-out: 0.2
- Recurrent activation function: Sigmoid
- Learning-rate: 0.0016
- Optimizer: RMSProp
- Batch normalization: True
- Layer normalization: False
- Use of momentum: False

The model is again provided with a heavy architecture: although having few layers, the number of LSTM cells per layer is here particularly important, i.e. 550 against 100/150 usually. Again, the heaviness of this model is due to the magnitude of the difficulty of the ASR task.

The drop-out has been slightly increased compared to the MLP model, this being due to the higher risk of overlearning in LSTMs due to their significantly higher complexity.

In LSTM networks, there is generally no use of activation functions between layers, as the cell computations, being largely based on the application of sigmoids and tanh, are considered sufficiently non-linear. We therefore did not have to add any additional activation functions at this level.

In order to ensure that the training is of sufficient duration for the model to have the opportunity to adapt to the complexity of the task it has to perform, we chose to lower the learning-rate, dividing it by 20 compared to the MLP training. This also has the advantage of avoiding local minima, since the update of the weights is less important at each gradient descent session, leading to a more progressive learning but also in theory more reliable.

For the optimizer, we chose again to follow PyTorch-Kaldi's recommendation on Librispeech, and thus to keep RMSProp, which, like SGD, is less used than Adam and its variants.

Moreover, similarly to previous trainings, we chose to favor normalization on batches rather than on model layers, as batch-normalization is known to have the following advantages:

- Improved training time and model accuracy
- Reduce model dependence to weights initialization
- It has a slight regularization effect

The results obtained during a training of 12 epochs are shown in Figure 22 and Figure 23.

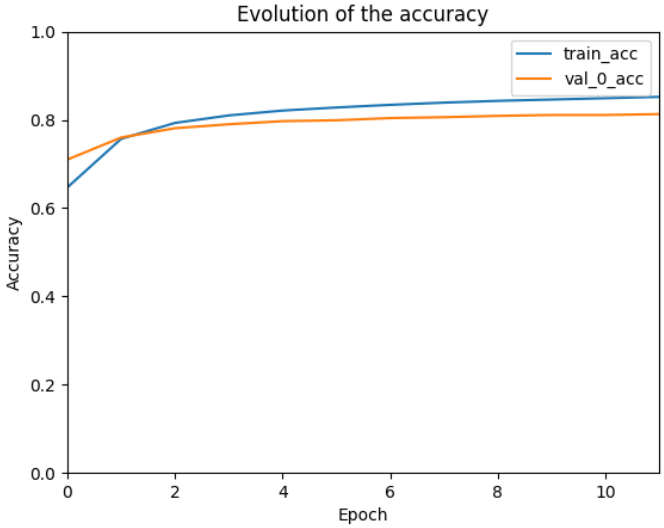


Figure 22: Accuracy for the LSTM model on MAILABS (12 epochs)

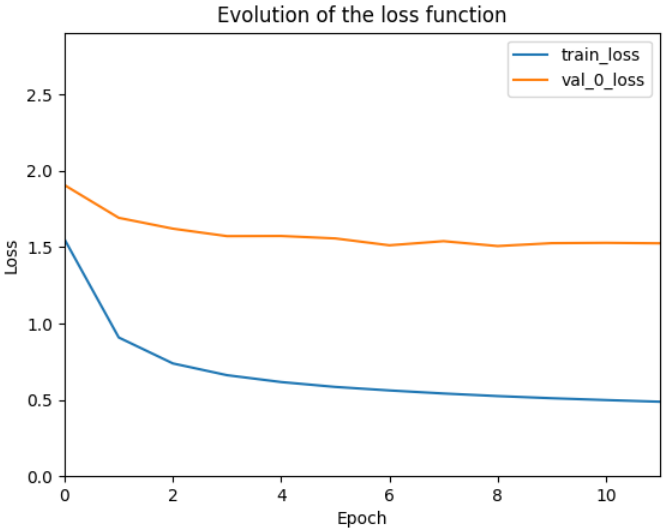


Figure 23: Loss for the LSTM model on MAILABS (12 epochs)

The following table shows the statistics of the model at different stages of its training:

Table 6: Results of Kaldi LSTM models on MAILABS

Epochs	1	5	12
Loss	1.551	0.617	0.488
Error	0.353	0.179	0.148
Validation loss	1.907	1.574	1.526
Validation error	0.290	0.203	0.187
Training runtime (min)	140	208	214
WER (%)	**22.54**	**16.92**	**16.27**

As we can see, compared to the MLP model for 1 epoch, the loss and error values are quite similar, but the WER is 1.50% higher and the training runtime is much higher (15 minutes for MLP vs 140 minutes for LSTM). Therefore, in this particular case, the MLP model outperforms the LSTM, which is quite surprising since LSTMs are usually more suited for NLP purposes thanks to its natural ability to capture recurrence in orderly sequences. This is in our theory a direct consequence of the reduced learning-rate: At this stage, weights of the model have simply not been updated enough to see the real benefits of the training. Indeed, from the 5th epoch, the trend was reversed and the LSTM model obtained significantly better results than the MLPs (WER of 16.92% against 19.08% on 5th epoch and 16.27% against 18.48% on 12th epoch).

The number of epochs, 12, is correctly chosen: at this point, the validation loss has stopped falling and has stabilised, which means that the model is no longer learning.

Moreover, we can see that, for 1 epoch, the losses and errors of the LSTM model are lower than the MLP model, but, as stated earlier, the WER is higher (for the LSTM). Our hypothesis for this problem is that the LSTM model being largely more complex than the MLP model, the latter can take care of a large part of the complexity of the problem, but at the same time making the associated decoding graph much less complex and thus finally obtaining a lower WER.

Again, while the WER performances aren't as good as LibriSpeech (10-11% for LibriSpeech vs 16% for MAILABS), they still can be considered as satisfying.

III.1.c) Difficulties encountered

Originally, we started with the installation of Kaldi on Windows. We managed to do the installation (the associated guide can be found here), but we couldn't build anything from it. Indeed, in general, ASR frameworks (in particular, PyTorch-Kaldi) are not compatible with other operating systems than Linux, but we weren't aware of that at the beginning of the POC phase. That's why, in this work, we mainly evolved on Linux environments (especially WSL2).

Moreover, the installation of Kaldi on Linux, which was supposed to be much simpler and faster than on Windows, turned out to be much more complex and time consuming than we thought. Indeed, many libraries and scripts were obsolete and/or corrupted, so we had to go through thousands of lines of Shell and Perl scripts in order to solve the associated issues.

Also, the installation of PyTorch-Kaldi (which, incidentally, is based on Kaldi's installation), didn't work at first because someone made a questionable pull request on Kaldi's Git in 2020, and we didn't realise it right away. Indeed, in that pull request, the author (of the request)

deleted the creation of the decoding graphs in almost all of the Kaldi-LibriSpeech executable, whereas we needed those graphs to perform the WER calculations and the rescoring process. Once the previous version of the Kaldi-LibriSpeech executable was recovered, everything started working properly.

Another limiting factor was the time it took to train the ASR models (MLP and LSTM) on PyTorch via WSL2. Indeed, at the beginning of the project, none of our workstations had an NVIDIA GPU, so we could not use CUDA (with PyTorch). Along the way, we were able to swap one of our workstations with another one, which had a model of NVIDIA GPU, series RTX2080ti, released in September 2018, and possessing a remarkable computing power (memory of 11GB for a capacity of operating at a frequency of 1350 MHz). Typically, on CPU, 1 epoch took about 3-4-5 hours to run. Using this kind of GPU, an epoch took only 15 minutes to complete! This allowed us to perform many more performance tests on PyTorch-Kaldi (for the MLP model at least).

III.1.e) Future Improvements

One of the biggest downsides of the pre-processing process via Kaldi is the execution time, especially during the creation of the decoding graph, and during the final decoding (via PyTorch-Kaldi) to obtain the WERs associated with the test phase (or the production phase).
One solution would be to use Online Kaldi, which would allow the decoding to be done via super-computers of the Online Kaldi framework, making the decoding almost instantaneous. This would potentially divide the runtime of the production phase by (at least) 2, and save 1 or 2 hours per test phase.

A second solution would be to use CUDA Kaldi. It's a modified version of the source code of Kaldi, such that almost all the calculations related to pre-processing are done via an NVIDIA GPU (assuming you have one) instead of the CPU, via CUDA. This would drastically reduce the execution time related to the pre-processing, while remaining on our local environment.
The idea is to be able to (ideally) combine the two previous solutions, in order to really optimise the execution time, in all the parts of our POC.

Another possible development of the project is to implement our own custom model on PyTorch-Kaldi (in the script neural_networks.py). This would allow us to test whether the MLP and LSTM models are indeed the most suitable and/or efficient in the scope of Kaldi and PyTorch-Kaldi.

In order to improve the WER, we could also use rescoring: this consists in correcting the predicted sentences with a language model (generally an n-gram model), and then re-running the WER computations on the new corrected sentences.

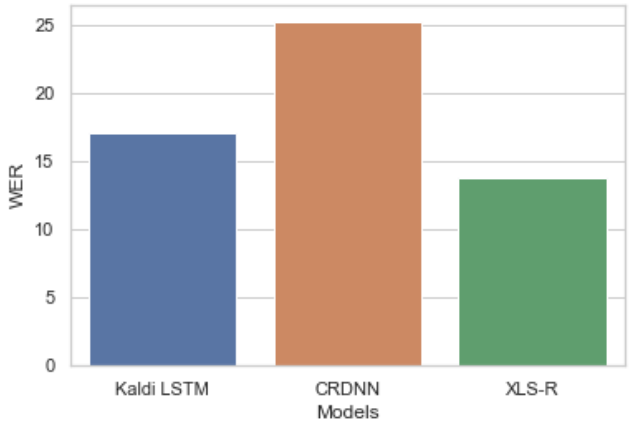
III.2) HuggingFace Models

III.2.a) Pretrained Models

We started the analysis of the performances of the HuggingFace models by evaluating their results on the test set of our French dataset, M-AILABS.

As a reminder, at this stage, neither of the two models, CRDNN and XLS-R, have been re-trained on M-AILABS, so this is a first step to evaluate the robustness of the performances of these networks.

The Figure 24 allows to compare them with the best model obtained with Kaldi/PyTorch-Kaldi, i.e. the model trained with an LSTM architecture on 12 epochs. Once more, the metric used is the WER. In addition, the associated table presents the performance and inference times of each of the models involved:



Model	Kaldi LSTM	CRDNN	XLS-R
WER (%)	16,27	25,27	13,74
Execution Time (s)	46,3	3,1	1,8

Figure 24: Model comparison on M-AILABS test

Through this example, we can realize the capacity of generalization of the best current ASR models: Even evaluated on a dataset never encountered before, the performances are interesting. The CRDNN, which is not considered state of the art, obtains a WER of 25.27%, which intuitively can be interpreted as "3 out of 4 words in the predicted text are correct".

Moreover, the XLS-R demonstrates its quality of phoneme modelling by obtaining a WER of 13.74%, a performance based on the robustness of its heavy pre-training. We can also observe that the best model obtained with Kaldi is surpassed, although it has been specially adapted on this dataset.

Finally, the execution times of the HuggingFace models are much lower (about 2 sec) than those which are the norm for Kaldi (about 50 sec). In the case of XLS-R, we can mention that the absence of preprocessing (no feature extraction, the calculations are made from the raw waveform) and decoding are determining factors in obtaining such a reduced prediction time.

To conclude, it seems that, even without re-training, the HuggingFace solution is more suitable for our use of the ASR model, both in terms of performance and execution time.

III.2.b) Fine-Tuned Model: XLS-R

In order to keep similar experimental conditions with the Kaldi training of the models, we rigorously kept the same samples for the fine-tuning of the XLS-R model. Thus, once again, the train set is composed of 30,000 audios and their transcriptions, against 3,000 and 3,000 respectively for the validation and test sets.

As the XLS-R is already a model dedicated to ASR, we did not have to modify its architecture. Moreover, due to lack of time, we did not try to implement the procedure of freezing some layers and to study the repercussions on the model performances.

The architecture of the model is thus identical to the original XLS-R model:

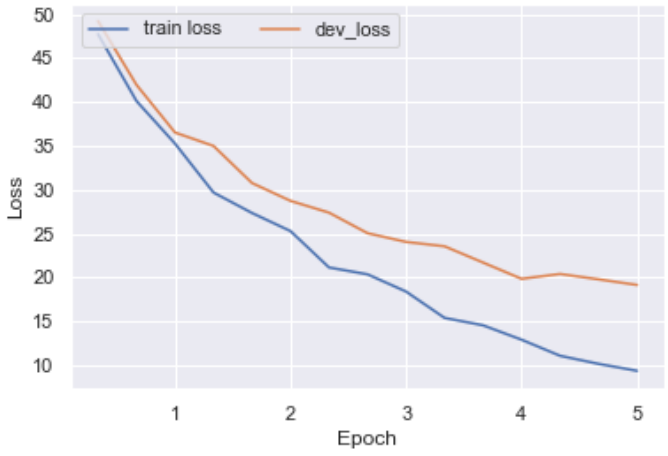
- Drop-out: 0.1
- Masking time probability: 0.05
- Hidden activation function: GELU
- Number of transformers layers: 1
- Number of convolutional layers: 7
- Number of convolutional filters: 7
- Number of hidden layers: 24

The drop-out value of 0.1 seems low for a model of this size and complexity. The authors chose to use an activation function halfway between the RELU and gaussian functions, i.e. GELU, which adds nonlinearity to the internal computations of the model. The probability that one value of the vector at the output of the convolutional block is hidden is 0.05, but as the next 10 values of the vector are also cancelled, there are more than 5% of hidden values in practice.

The hyperparameters chosen for the training are the following:

- Number of epochs: 5
- Evaluation steps: 5000
- Training batch size: 4
- Learning rate: $3e^{-4}$
- Optimizer: AdamW

As this is a first test and the fine-tuning is supposed to be a slight touch-up of the model weights, we chose to reduce the number of epochs to 5. The learning-rate is relatively low, in order to guarantee a good quality training. Because of memory limitations on our GPUs, we were forced to set a very low batch size, here 4, which means that the computation of the gradients will be done iteratively after using only 4 audio samples. Moreover, HuggingFace being much more recent than Kaldi, we were in a position to use an optimizer more powerful than SGD and RMSProp, that is to say the algorithm AdamW [36], a weight decay corrected version of the reliable Adam. The results of the training are illustrated in Figure 25:



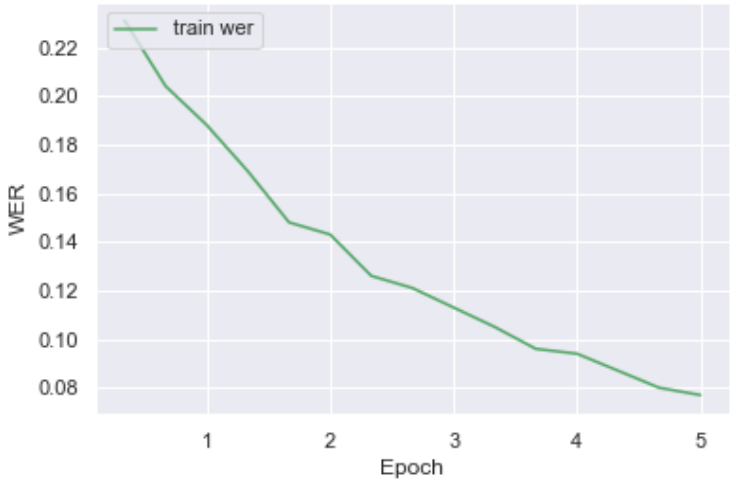


Figure 25: XLS-R Finetuning results

On this example, we notice that the training takes place under favorable conditions: The training and validation losses decrease rapidly. In response, as expected, the training WER also decreases between epochs, with a value of 7.72% at the end of the 5th epoch. On the test dataset, the WER obtained is 11.43%, which places the XLS-R fine-tuned as our best model in terms of raw performance.

However, unfortunately, the number of epochs appears to be too small: at the end of the 5th epoch, the validation loss is still far from being stabilized and continues to decrease at a steady pace. In response to this phenomenon, we envisaged the solution of launching a second training, increasing the learning-rate (3e-4 from to 3e-3) and increasing the number of epochs, with an early stopping mechanism that allows us to stop the training as soon as the difference in validation loss between two epochs is below a certain threshold (0.01). Yet, the time constraint was too great and, having several training sessions in progress, we were unable to free up the necessary resources for this training, unfortunately postponing it to a date later than this report. Its results are therefore not currently available.

III.2.b) Difficulties encountered

The HuggingFace platform is a great tool for people interested in NLP: it offers the possibility to reuse pre-trained models or royalty-free datasets at will. However, as many of the functions that handle these options are already implemented, it can be complex to get familiar with the platform environment.

In particular, the library has specific classes for holding dataset samples and requires their use in any interaction between a platform model and a dataset. Thus, we had to format M-AILABS in such a way that it was saved in an instance of the Dataset class, this was not without difficulty because loading audio files (.wav in our case) must be done in a particular way, and tutorials on this subject are noticeably lacking. In particular, we first encountered large memory allocation problems and in a second time backend problems for loading the audio.

Furthermore, similar to what we had encountered during the Kaldi experimentation phase, we were confronted to a lack of computing power for the model training phase. Having only one GPU, which after some tests seemed insufficient to train a model as large as the XLS-R, we first tried to use the resources available in the cloud, through the Amazon training platform called "SageMaker". However, each of the instances we used led to CUDA out of memory issues and by studying the technical specifications of Amazon's GPU Tesla T4, we realised that Aubay's model, RTX 2080ti, is far more powerful (1350 MHZ clock speed for 2080ti vs 1005 MHZ for Tesla T4). We therefore went back to our original solution, by reducing the training batch size from 32 to 4 and the precision of the floats used from 32 to 16, thus avoiding out-of-memory errors, and finally trained the model locally.

III.3) Further information: Agile methodology

At the beginning of the 2022 spring session, Aubay Innov' supervisors have chosen to alter the pace of work by applying the principles of a particular methodology called "Agile".

"Agile Methodology" can be defined as a way of organizing projects allowing great flexibility, and a lot of adaptability. Thus, the organization of the project can be adapted, and the needs can simply be changed. The Agile organization opposes the classical V-cycle, where all the specifications of the product are defined before developing the entire project, and known to the customer uniquely at the end of the development.

With Agile mode, we determine "a sprint", that is to say a certain number of weeks (usually between 1 and 4), during which the developers work on the tasks assigned individually to them. On FYW, we have chosen to set the duration of the sprints at 2 weeks, to have enough time to react in case a task poses a problem. Once the sprint is finished, the work carried out is reviewed, in order to identify possible areas for improvement, and we determine the tasks to be done in the next sprint (backlog).

In addition to the organization in sprint, the agile imposes the holding of daily meetings, and not weekly as during the first part of the work-study year. This allowed to amplify in a considerable way the communication between the members of the team, since each one knew exactly the task assigned to the other developers, which helped to establish a systematic mutual aid among the members of the team.

Conclusion

In this first POC of the NLP section of the "Find Your Way" project, and with the help of my team at “Aubay Innov’”, I had the occasion to investigate different “Automatic Speech Recognition” solutions.

Most of our work has been done with the Kaldi framework, which is considered as a must-try by the ASR research community. The installation of the framework proved to be extremely complex, some scripts being runnable only under Linux, which forced the use of WSL2, the second version of the ubuntu kernel for windows. Moreover, the codes consisting of the official Kaldi (and PyTorch-Kaldi) tutorial were not directly usable on our PCs and mostly required many modifications. However, once these adaptations became effective, we were able to perform adequate preprocessing (Phonemizer, lexicon, language model, speaker identification documents, ...), to launch trainings of ASR models (MLP and LSTM architectures mainly) and to put them in production. The performance of our models exceeds our expectations, since we obtain a WER of less than 20%, which, even compared to state-of-the-art models (XLS-R / CRDNN), is relatively impressive. However, as a trade-off for these performances, we have observed a high execution time that hinders the implementation of a real-time system: indeed, most of our Kaldi-trained models have prediction times of more than 30 seconds. It seems that solutions exist for this problem (Online Kaldi, CUDA Kaldi, etc), which would be one of the main ways to improve this POC.

In a second step, we were interested in the use of pre-trained models, freely available on the Huggingface platform. The two models evaluated, namely a French version of a recent model from the well-known Wav2Vec2 proposed by Facebook AI, and a CRDNN-type model, obtained disparate performances with respectively 13% and 25% of WER on M-AILABS (French ASR dataset used during the project). These results, obtained on a dataset not used during their training, prove the robustness of the models. Moreover, the prediction time of these two models is reduced, the transcription being available in less than 3 seconds on average. In a second step, we were able to apply a fine-tuning procedure to the XLS-R model (re-training with a different dataset), which allowed us to use the full potential of the network on the M-AILABS dataset and to bring its WER performance to 11% in a reduced number of epochs.

Tables

Table of figures

Sources of figures

- First Page Illustration: https://colab.research.google.com/drive/1fXJ_YUWACs0w8ohORFijuWo4_pCLMIEK
- "Université de Paris"'s logo : https://fondation-uparis.org/2022/03/16/universite-paris-cite-un-nouveau-nom-pour-notre-universite-et-sa-fondation/ (Illustration cropped)
- Aubay's logo (figure 1): https://blog.aubay.com/index.php/language/en/home/
- Options from Kaldi (figure 4): https://developer.nvidia.com/blog/gpu-accelerated-speech-to-text-with-kaldi-a-tutorial-on-getting-started/ (Illustration cropped)
- Characters added to M-AILABS (figure 5): https://www.caito.de/2019/01/03/the-m-ailabs-speech-dataset/
- Three-Layers MLP (figure 7): https://medium.com/analytics-vidhya/in-depth-tutorial-of-recurrent-neural-network-rnn-and-long-short-term-memory-lstm-networks-3a782712a09f (Illustration annotated)
- Memory cell of a LSTM model (figure 8): https://www.altoros.com/blog/text-prediction-with-tensorflow-and-long-short-term-memory-in-six-steps/ (Illustration cropped)

- Representation of WFST with 6 states and 8 transitions (figure 9) : https://www.researchgate.net/publication/331894180_Prise_en_main_du_KaldiWFST_Toolkit
- Pytorch-Kaldi architecture (figure 10): https://arxiv.org/abs/1811.07453
- Wav2Vec2.0 architecture (figure 12): https://towardsdatascience.com/wav2vec-2-0-a-framework-for-self-supervised-learning-of-speech-representations-7d3728688cae?gi=3631dffb8323
- XLS-R comparison with ASR models (figure 13): https://arxiv.org/pdf/2006.13979v2.pdf
- XLS-R architecture, similar to Wav2Vec2.0's (figure 14): https://arxiv.org/pdf/2111.09296.pdf
- Comparison between XLS-R models and previous ASR works (figure 15): https://arxiv.org/pdf/2111.09296.pdf

Table of tables

Bibliography

Research articles consulted

SER Datasets

[1] Livingstone SR, Russo FA (2018) The Ryerson Audio-Visual Database of Emotional Speech and Song (RAVDESS): A dynamic, multimodal set of facial and vocal expressions in North American English. PLoS ONE 13(5): e0196391. https://doi.org/10.1371/journal.pone.0196391.

[2] Gournay, Philippe & Lahaie, Olivier & Lefebvre, Roch. (2018). A canadian french emotional speech dataset. 399-402. 10.1145/3204949.3208121.

Other PoCs main algorithms

[3] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 779-788, doi: 10.1109/CVPR.2016.91.

[4] Chien-Yao Wang, I-Hau Yeh, and Hong-Yuan Mark Liao, "You Only Learn One Representation: Unified Network for Multiple Tasks", 2021, arXiv:2105.04206v1

[5] Carlos Campos and Richard Elvira and Juan J. Gomez Rodriguez and Jose M. M. Montiel and Juan D. Tardos, (2020), "ORB-SLAM3: An Accurate Open-Source Library for Visual, Visual-Inertial and Multi-Map (SLAM)", CoRR, https://arxiv.org/abs/2007.11898

State of the Art: Automatic Speech Recognition

[6] Povey, Daniel & Ghoshal, Arnab & Boulianne, Gilles & Burget, Lukáš & Glembek, Ondrej & Goel, Nagendra & Hannemann, Mirko & Motlíček, Petr & Qian, Yanmin & Schwarz, Petr & Silovský, Jan & Stemmer, Georg & Vesel, Karel. (2011). The Kaldi speech recognition toolkit. IEEE 2011 Workshop on Automatic Speech Recognition and Understanding.

[7] Ravanelli, Mirco & Parcollet, Titouan & Bengio, Y.. (2018). The PyTorch-Kaldi Speech Recognition Toolkit, ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), DOI:10.1109/ICASSP.2019.8683713

[8] Davis, S.V. & MERMELSTEIN, PAUL. (1980). Comparison of Parametric Representations for Monosyllabic Word Recognition in Continuously Spoken Sentences. IEEE Transactions on Acoustics, Speech and Signal Processing. 28. 57-366. 10.1016/B978-0-08-051584-7.50010-3.

[9] Viikki, Olli & Laurila, Kari. (1998). Cepstral domain segmental feature vector normalization for noise robust speech recognition. Speech Communication. 25. 133-147. 10.1016/S0167-6393(98)00033-8.

[10] N. Dehak, P. J. Kenny, R. Dehak, P. Dumouchel and P. Ouellet, "Front-End Factor Analysis for Speaker Verification," in *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 19, no. 4, pp. 788-798, May 2011, doi: 10.1109/TASL.2010.2064307.

[11] Panayotov, Vassil & Chen, Guoguo & Povey, Daniel & Khudanpur, Sanjeev. (2015). Librispeech: An ASR corpus based on public domain audio books. ICASSP 2015 - 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). 5206-5210. 10.1109/ICASSP.2015.7178964.

[12] Jodi Kearns (2014), "LibriVox: Free Public Domain Audiobooks", Reference Reviews, Vol. 28 No. 1, pp. 7-8. https://doi.org/10.1108/RR-08-2013-0197

[13] Shannon, Claude. (1948). A Mathematical Theory of Communication. 10.7551/mitpress/12274.003.0014.

[14] Kenneth Heafield. 2011. KenLM: Faster and Smaller Language Model Queries. In Proceedings of the Sixth Workshop on Statistical Machine Translation, pages 187–197, Edinburgh, Scotland. Association for Computational Linguistics.

[15] B. Atal and M. Schroeder, "Predictive coding of speech signals and subjective error criteria," in *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 27, no. 3, pp. 247-254, June 1979, doi: 10.1109/TASSP.1979.1163237.

[16] Ravikumar KM, Rajagopal R, Nagaraj HC. An approach for objective assessment of stuttered speech using MFCC features. ICGST International Journal on Digital Signal Processing, DSP. 2009;9(1):19-24

[17] M.J.F.Gales, 1998, "Maximum likelihood linear transformations for HMM-based speech recognition", Cambridge University Engineering Department, CUED/F-INFEND/ TR 291

[18] Baum, L. E.; Petrie, T. (1966). "Statistical Inference for Probabilistic Functions of Finite State Markov Chains". The Annals of Mathematical Statistics. 37 (6): 1554–1563. *doi:10.1214/aoms/1177699147*

[19] John S Garofolo and Lori F Lamel and William M Fisher and Jonathan G Fiscus and David S Pallett and Nancy L Dahlgren (1993). DARPA TIMIT: (Technical report). National Institute of Standards and Technology. *doi:10.6028/nist.ir.4930*

[20] Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, *65*(6), 386–408. https://doi.org/10.1037/h0042519

[21] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," in *Neural Computation*, vol. 9, no. 8, pp. 1735-1780, 15 Nov. 1997, doi: 10.1162/neco.1997.9.8.1735.

[22] Mohri, Mehryar. (2000). Finite-State Transducers in Language and Speech Processing. Comput Linguist. 23.

[23] Mohri, Mehryar. (2004). Weighted Finite-State Transducer Algorithms An Overview. Formal Languages and Applications. 148. 10.1007/978-3-540-39886-8_29.

[24] Devlin, Jacob & Chang, Ming-Wei & Lee, Kenton & Toutanova, Kristina. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.

[25] Vaswani, Ashish & Shazeer, Noam & Parmar, Niki & Uszkoreit, Jakob & Jones, Llion & Gomez, Aidan & Kaiser, Lukasz & Polosukhin, Illia. (2017). Attention Is All You Need.

[26] Baevski, Alexei & Zhou, Henry & Mohamed, Abdelrahman & Auli, Michael. (2020). wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations.

[27] Conneau, Alexis & Baevski, Alexei & Collobert, Ronan & Mohamed, Abdelrahman & Auli, Michael. (2021). Unsupervised Cross-Lingual Representation Learning for Speech Recognition. 2426-2430. 10.21437/Interspeech.2021-329.

[28] Wang, Changhan & Tjandra, Andros & Lakhotia, Kushal & Xu, Qiantong & Goyal, Naman & Singh, Kritika & Platen, Patrick & Saraf, Yatharth & Pino, Juan & Baevski,

Alexei & Conneau, Alexis & Auli, Michael. (2021). XLS-R: Self-supervised Cross-lingual Speech Representation Learning at Scale.

[29] Ardila, Rosana & Branson, Megan & Davis, Kelly & Henretty, Michael & Kohler, Michael & Meyer, Josh & Morais, Reuben & Saunders, Lindsay & Tyers, Francis & Weber, Gregor. (2019). Common Voice: A Massively-Multilingual Speech Corpus.

[30] Roach, Peter & Arnfield, Simon & Barry, William & Baltova, J. & Boldea, Marian & Fourcin, Adrian & Gonet, Wiktor & Gubrynowicz, Ryszard & Hallum, E. & Lamel, Lori & Marasek, Krzysztof & Marchal, Alain & Meister, Einar & Vicsi, Klara. (1996). BABEL: An Eastern European multi-language database. International Conference on Spoken Language Processing, ICSLP, Proceedings. 3. 10.1109/ICSLP.1996.608002.

[31] Martin, Louis & Muller, Benjamin & Ortiz Suarez, Pedro & Dupont, Yoann & Romary, Laurent & De la Clergerie, Eric & Seddah, Djamé & Sagot, Benoît. (2019). CamemBERT: a Tasty French Language Model.

[32] Wolf, Thomas & Debut, Lysandre & Sanh, Victor & Chaumond, Julien & Delangue, Clement & Moi, Anthony & Cistac, Pierric & Rault, Tim & Louf, Rémi & Funtowicz, Morgan & Brew, Jamie. (2019). Transformers: State-of-the-art Natural Language Processing.

[33] Ravanelli, Mirco & Parcollet, Titouan & Plantinga, Peter & Rouhe, Aku & Cornell, Samuele & Lugosch, Loren & Subakan, Cem & Dawalatabad, Nauman & Heba, Abdelwahab & Zhong, Jianyuan & Chou, Ju-Chieh & Yeh, Sung-Lin & Fu, Szu-Wei & Liao, Chien-Feng & Rastorgueva, Elena & Grondin, François & Aris, William & Na, Hwidong & Gao, Yan & Bengio, Y.. (2021). SpeechBrain: A General-Purpose Speech Toolkit.

[34] Allauzen, Cyril & Riley, Michael. (2007). OpenFst: A general and efficient weighted finite-state transducer library.

[35] Kingma, Diederik & Ba, Jimmy. (2014). Adam: A Method for Stochastic Optimization. International Conference on Learning Representations.

[36] Ilya Loshchilov and Frank Hutter, (2019), Decoupled Weight Decay Regularization. International Conference on Learning Representations, https://openreview.net/forum?id=Bkg6RiCqY7

Additional Websites consulted

[37] Hart, Michael S, (1974), "The Gutenberg Project" https://www.gutenberg.org/

[38] Imdat Solak (2019), "The M-AILABS Speech Dataset" https://www.caito.de/2019/01/03/the-m-ailabs-speech-dataset/

[39] Jonathan Duddington, Reece Dunn, (2006), "eSpeak NG Text-to-Speech" https://github.com/espeak-ng/espeak-ng/

[40] Jonathan Fiscus, (2021), "SCTK, the NIST Scoring Toolkit" https://github.com/usnistgov/SCTK