Université Toulouse III - Paul Sabatier
118 route de Narbonne
31062 TOULOUSE

Lincoln Centre for Autonomous Systems, University of Lincoln, Brayford Pool, Lincoln, LN6 7TS, United Kingdom

A simple ASR model for robot command recognition

An end-to-end speech and language recognition model for robot commands

April 2021 – July 2021

Presented and supported by

Aurélien Marchal

Thanks

I want to thank Dr.Cuayahuitl for helping me in the project, Dr. Ferrane for giving me advice in the field of speech recognition and inviting me at the Irit lab, and finally Abdel Heba for his course on Speechbrain at the Irit Lab.

Index

I. Context

This internship was conducted with the Lincoln Centre for Autonomous Systems of the University of Lincoln, UK, under the supervision of Dr. Cuayahuitl. Due to the COVID-19 pandemic, the entire internship was conducted remotely from France.

The subject is parallel to the subject of Cederick Moulliet, also intern at the LCAS and also under the supervision of Dr. Cuayahuitl. He worked on robots manipulation tasks such as picking up objects for the Pepper Robot. My work was to build a speech recognizer for the robot, so that the robot can be controlled by voice to perform the tasks that Cederick implemented. Although his researches are very influenced by the abilities of the Pepper Robot, mine are not meant to be integrated on the robot for now and thus the abilities will not be taken into account.

All the code I wrote can be found in this Gihub repository : https://github.com/AurelienMarchal/Speech-to-Command . For the entire report, every file mentioned is present in the repository (to avoid having the entire code in annex). I will put the name of the file in brackets ([train.py] for example).

I will be dividing my report in 3 main parts. In the first one, I will introduce the subject and the state of the art related, as well as present both my scientific and technological methodology. In the second part, I will talk about the work done, my choices and the results I got. In the 3rd part I will present how the model can be used and I will possible improvements to my work. A last and final part will be dedicated to my personal experience and what remains to be done.

II. Subject presentation

II.1 Problematic

II.1.a) Technological issues

Speech recognition for robots is definitively not a new problem. Since the development of Neural Networks, speech recognizers have improved tremendously and are still being perfected to this day. They are a key part of Human/Machine interactions and they help us communicate effortlessly to robots in a natural way. They play a big part in integrating robots and AI assistants in our everyday life.

However, as speech recognizer technologies develop, datasets are getting bigger and NN are getting more complex and costly to train. On top of that, understanding a human talking is not as simple as building an ASR (Automatic Speech Recognition) model (a model is a Neural Network that has been trained), the machine needs to understand the meaning of the sentence. This is called Intent Recognition or language interpretation and it is a different problem in itself. Intent Recognition is also solved by complex NN that also require lots of data to train.

II.1.b) Goal

The aim of this internship is to build an end-to-end (meaning without the help of another service) Speech Recognizer without using huge datasets. Everything must be done without using already built ASR models of Intent recognitions models or even widely used datasets. Furthermore, the speech recognizer is meant to be implemented for personal assistant robots, like the Pepper Robot, that do not have big computing power, so the model needs to remain simple. This also means that the vocabulary needs to be adapted to robot tasks. To resume, the model needs to recognize and understand commands and orders for the robot from the voice of the user.

II.1.c) Limitations

The biggest limitation for this subject is the lack of a big dataset. Indeed, it is impossible to make an ASR model that could work for all users, each having their own voice and accent, without having a huge dataset with recordings of people with different accents and pitch. In order to have a simple but efficient model, it needs to be specific for each user. Thus, the user will have to record a small dataset for the model to recognize its voice otherwise this subject is not possible.

For the same reason, the language register must be very basic because language interpretation of natural speech is not a trivial problem to solve. It requires a lot of knowledge of the possible turn of phrases and grammatical structure of the language.

II.2 Current technologies

II.2.a) State of the art

Extracting actions to perform from speech is almost always solved in two steps: Speech to Text (ASR) and language interpretation.

The state of the art ASR are end-to-end, seq-to-seq models using RNN (Recurrent Neural Networks) as presented by the Amazon Alexa researchers [1] or by the Google Team [2]. The Facebook AI researchers recently came up with an unsupervised model for feature extraction called wav2vec [3] that appears to out-perform current ASR models with less labelled data. These models are often developed on datasets such as Timit [4] or Librispeech [5].

For language interpretation, deep reinforced learning is often used [6]–[8] for robot tasks. Seq-to-Seq NLP (Natural Language Processing) model are also used [9] [10] for robust understanding of robot directed commands.

II.2.b) Scale problem

Recreating those types of models is a big challenge because they are often very complex and require a very deep understanding of ASR technologies. Moreover, this technologies are developed by GAFAM companies labs that have almost unlimited resources in term of computing power and the best researchers of the domain. However, Google and other companies give access to their ASR models via API witch is very practical but not the goal of this internship. Also, if I want to reuse this approaches, it will mean that I will have to develop two separate models when building only an ASR model from scratch is already a big challenge for my limited resources.

II.2.c) Simplifications

Some simplification were made to make this project possible.

First, I choose commands with syntax so simple that I did not need an NLP model. This will be discussed more into details in III.1.a . That way, I only had to focus on building an ASR model. The big downside is that users will not be able to use natural language but only pre-defined commands.

Secondly, the ASR model will be meant to recognize only the user who recorded the dataset. This removes the constraint of having to recognize every type of voice and accent, preventing me from requiring a lot data from multiple speakers. This will also means that a user will have to record himself before using the speech recognizer, whitch is far from ideal, but is unavoidable for this approach. Thus the recording time of the dataset must be kept to a minimum.

Finally, as I can not have the user record a lot of data, the number of different commands and variation of commands will be small (less than 5 commands with less than 50 variations for each).

In part IV, I attempt to minimize those restrictions to make the model more applicable in a real context.

II.3 Scientific Method

The method I used was to start with simple concepts, get them working and then later, improve the model and start generalizing the problem with a more sophisticated approach. This method was chosen because the internship only lasted 3 month and an important aspect of this internship was that I built almost everything by myself, including the dataset. I was still able to use Python libraries but I did not use existing dataset or "pretrained" models. This was more challenging than just improving an existing recipe, but it gave me the freedom to build the project as I wanted.

The internship was composed of 3 phases, each having its own requirements. The technical details will be mentioned in part III.

II.3.a) Record a small dataset

The first step of the internship was to build a small-scale dataset for robot commands. It was important to start from the dataset because the recipe, especially the data pipeline part, will greatly depend on how the dataset is built. Also, there is no universal way to build a dataset, it depends on the problem to solve, so building a dataset already gives an orientation on the architecture of the model.

An important aspect of the project was that the user could record its own dataset. Thus, the dataset had to be quick and easy to record to take that requirement into account.

In addition, the vocabulary had to be restricted because the model had to be simple. Also, a larger vocabulary would lead to a bigger dataset which was not the goal.

II.3.b) Experimenting with model architectures

After I built the dataset, I focused on trying different model architectures. Building the architecture of a model amounts to choosing which layer of neural network (NN) to stack on top of each other, as well as the inputs and outputs of the NN. This requires previous knowledge on the different types or NN layers to know what will possibly work.

One important thing to take into account was the training time together with the computing power required. The model had to be fast to train (< 10 mins) on a small GPU.

The results did not have to be great initially, they would be improved in the last part of the internship.

II.3.c) Improve the accuracy/model

Once I was satisfied with the model, I improved its performances in any of the following way: improve the accuracy, lower the training time, lower the computing power required, decrease the number of recording while keeping the same accuracy, lower the prediction time ··· depending on the biggest flaws of the model.

Before getting in the actual realisation, I need to mention the main technical framework I used during the whole internship.

II.4 Technical Method

I only used Python for coding. Python is the most used language in machine learning by far and it is very easy to use and permissive.

For the most part, I followed the SpeechBrain template to make the recipes (a recipe is a set a script that can be ran along with a dataset to build a model).

II.4.a) SpeechBrain

SpeechBrain is a Python library specificity made for speech related NN. This library was recommended to me by Dr. Ferrané and I had the chance to be invited to the IRIT laboratory for an 8 hour formation on the library by Abdel Heba, who is one of the main contributor. It is very recent, it was created in 2021 by Dr. Mirco Ravanelli and Dr. Titouan Parcollet. It is a NN framework only made for Speech Recognition [11](Speaker Recognition, ASR, Intent Recognition, Translation) and is very practical to use for many reasons. https://speechbrain.github.io/index.html

First, SB is built on Pytorch, which is the most powerful and used NN library in the world along with Tensorflow (Pytorch is made by Facebook and Tensorflow is made by Google, both have their pros and cons [12]). This means that it benefits from the power of Pytorch and all the predefined support that comes with it.

In addition, the major benefit of using SpeechBrain is the clarity of the recipe and the structure of the code. All the parameters and the layer definitions go to a separate file called the *hyperparameter* file and that prevents the user from hardcoding values directly in the Python witch is not good practice. This also means that a NN could be train with several parameters depending on the *hyperparameter* file used. There is a lot of other code optimization and built-in tools but I will not mention all of them.

Finally, for researchers, SpeechBrain provides the state-of-the-art recipes for the main datasets. This gives an idea on how SpeechBrain recipes should be written as well as the architecture of the state-of-the-art models. This feature helped me a lot through out my research.

II.4.b) Github

I used Github even though I was doing this project by myself because I wanted to get more comfortable with this tool. It helped me transfer my code to Google Colab easily and in general just share and show my code easily.

The link to the Github is : https://github.com/AurelienMarchal/Speech-to-Command

In there you will find the entire code, along with a Readme that contains instruction to execute the code. If I mention a file that is not in annex, it will be present in the repository.

I wrote a Google Colab Notebook for everyone to use if someone wants to see the training process without having to install anything.

III. Realisation

III.1 Building the Dataset

III.1.a) Corpus

The entire list of commands is the file [full_command_list.txt]. The commands are composed of 4 different elements: command | object 1 | preposition | object 2. This was inspired by [8] where each action is described by elementary sub-action made out of action, object1, preposition, object2. This very simple syntax still allows to describe plenty of action that a robot could do: "place the cup on the table", "clean up the table with the sponge", "set an alarm at 12:30" etc ···. Obviously, this type of language is not very natural but it allowed me to skip the intent recognition part and go directly from speech to intent. If the commands are always in the same form, we do not require a model to detect that the first word is the action/command, the second is the object1 and so on. Furthermore, the words can be bigrams joined together for simplicity. For example, the bigram "left of" becomes one single preposition "left_of which be treated as a single word. The command "clean up the table with the sponge" can be interpreted as : {command: "clean_up", object1: "the_table", preposition: "with", object2: "the_sponge"}. This allows for some almost correct English as it would not be natural to say "clean table with sponge".

In the case of this experiment, the speech model is meant to work on a robot for object manipulation. For now, the objects are replaced with coloured cubes just like in [13]. To avoid repetition and useless words, "red cube" was replaced by "red" and so on with blue, green and yellow. Then, I choose some action to perform on those cubes and some preposition to add some variation to these action. The [command.json] file describes every preposition that goes with each command and whether a command needs an object or not. The command “pick up” gives an example that a command could only be made of command | object1 or command | preposition if we choose to leave one field blank. I did not ended up including commands with empty fields but that could be an improvement for the future.

The entire corpus is composed of 108 possible commands but with only 15 different words of vocabulary. That way, a word of vocabulary is repeated several times during the recording, thus increasing the number of time the NN will be train to recognize it.

put	72
red	54
blue	54
green	54
yellow	54
move	24
on	24
far_from	12
near	12
stack	12
bellow	12
left_of	12
right_of	12
in_front_of	12
behind	12

Figure 1: Utterance of each word in the 108 possible commands

III.1.b) Recording

For the recording process I wrote a script [populate_dataset.py] that goes though the entire list of commands [full_command_list.txt] and let the user record each command a given number of times. In file [populate_dataset.py], line 26, this number is stored in the variable `nb_iteration`. I choose 5 for its value to have 540 recordings which is enough to work with for now.

To get the utterance of each word in the final dataset, we simply need to multiply by 5 the utterances in Figure 1.

Again, in file [populate_dataset.py], on line 17 through 20, the audio parameters are set, such as sampling frequency, number of channels (mono or stereo) etc ···. This values are very common for speech recognition with NN usage like in [4], [5], [14].

The script gives the user 5 seconds to record each command. Once the command is recorded, the script will cut out the silent parts by using V.A.D. [15](Voice Activity Detection) from [16]. Then the recording are stored in .wav files in a folder named after the speaker name. That way, the recordings of another speaker would be sorted separately for the other ones for convenience.

III.1.c) Annotations

Building a dataset does not only amounts to recording speech and storing them. Indeed, the dataset needs to contain informations relative to the files, called annotations. The annotation file needs to, at least, contain the transcript of every entry in the dataset, otherwise the NN will not be able to train. It can also contain informations such as the speaker identity, the duration of the file, the intent of the transcript and more, depending on the usage of the dataset.

I followed what was recommended in the SpeechBrain tutorials and stored the annotations in a dictionary structure in a JSON file. Every entry has a unique id and contains the file path, the speaker name, the duration and the transcript. Here is an example:

```
"stack_green_on_red_515mzgwfq0": {
        "full_command": "stack green on red",
        "wav": "./dataset/wavs/aurelien_marchal/stack_green_on_red_515mzgwfq0.wav",
        "duration": 1.92,
        "speaker": "aurelien_marchal",
        "cmd": "stack",
        "obj1": "green",
        "prep": "on",
        "obj2": "red"
    }
```

The dataset contains a total of 20 minutes and 12 seconds of audio for 540 files with one speaker and took about 4 hours to record entirely. Each possible command was recorded 5 times. Compared to a widely used dataset like TIMIT [4] which has a total of 5.4 hours of recording, this is a very small one but building a dataset like TIMIT took years of recording.

III.2 Data processing

III.2.a) Data pipeline

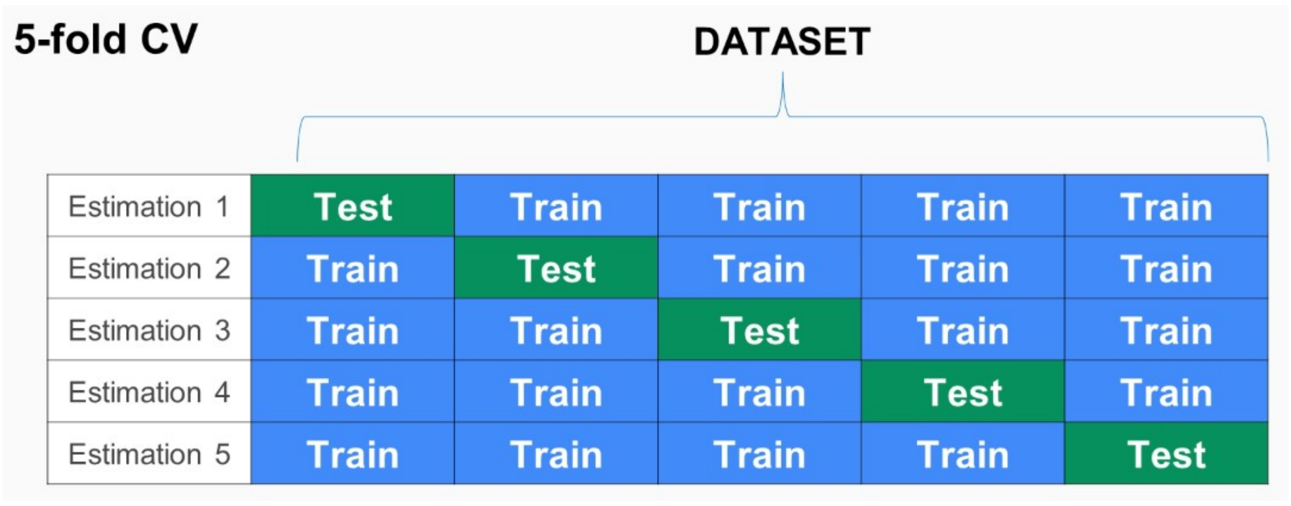
The data pipeline refers to all the processing applied on the data between the dataset files to the input of the NN. It can also include the formatting of the labels because Neural Networks do not output words directly, they output a number corresponding to a word or a class or a phoneme etc ⋯.

All of the data pipeline is defined in the dataio_prepare function in the file [train.py]. First, the signal is extracted from the audio file (line). Then a label encoder is created giving each word a number as showed in the file [label_encoder.txt]. The <bos> and <eos> words stand for "beginning of sentence" and "end of sentence" . This words mark the start and end of every sentence and are needed later by the encoder decoder so it is important to add them.

III.2.b) Data augmentation

Data augmentation a crucial part of any NN training, especially when the dataset is small. Its goal is too enhance the model performances and capacity of adaptation without increasing the size of the dataset. This can be archived by artificially increasing the number of recording or adding fake noise to the recording to train the NN to get used to potential noise in the future recordings to predict.

On one hand, I used cross validation with k-folding to be sure to use every recording to train, valid and test the model. This method is often used when the number of recording is small [17]. It consist of going through the data multiples times (k times) but changing what part of the data set is used for training and validating (or testing). This make sure every recording is used for testing at some point in the training loop if we use the maximum folding number. If, for example, we use 80% of the data for training and 20% for testing, we would have a maximum of 5 folds to use the whole dataset as testing.



5-fold CV

DATASET

Estimation 1	Test	Train	Train	Train	Train
Estimation 2	Train	Test	Train	Train	Train
Estimation 3	Train	Train	Test	Train	Train
Estimation 4	Train	Train	Train	Test	Train
Estimation 5	Train	Train	Train	Train	Test

Figure 2: 5 fold data structuring

In my case it is a little different since I cut the dataset in 3 parts: train, valid, test; but the idea is the same. My dataset has 5 utterances of each recording. Thus, in the file [prepare.py], I can assign 3 recordings to training, 1 for validating and 1 for testing (60%, 20%, 20%) for each command. I repeat this for every fold and with 5 folds, the entire dataset will used at leat one time for training and validating.

On the other hand, I used two approaches that are widely used in speech recognition. They directly modify the recordings to artificially increase the number of data and to improve the robustness of the model.

The first method is more straightforward. I used the built-in data augmentation class of SpeechBrain to vary the speed of each recording. This method produces 3 recordings from one original, one where the speed is set to 95%, one to 100% (no change) and one to 105%. This way, the NN will train on faster and slower recording which will make it more robust to slower and faster speakers.

The last method consists of adding fake noise and reverb to simulate different environments of recording. Varying the reverb artificially changes the size of the room and adding noise mimic potential parasite sounds surrounding the user (radio, car sound, street sound, TV ⋯)[18].

III.3 Model Architecture

III.3.a) Wav2vec with convolutional NN 1st approach

The first idea was to use wav2vec for feature extraction and a regular model on top. Wav2vec is an unsupervised NN that extracts features (or vectors) from raw audio. [3] shows that a model trained with this technology reaches the same Word-Error-Rate as the state-of-the-art models but with less labelled data. We can then feed this features to a regular NN (often an Encoder Decoder) to get the desired outputs.

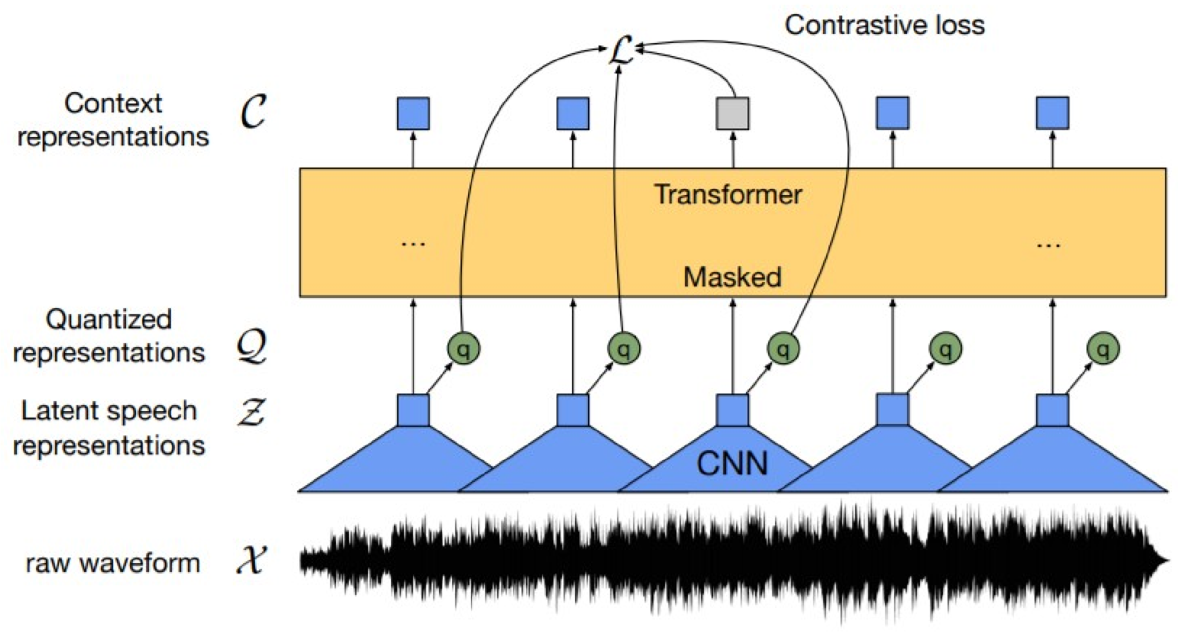


Figure 3: wav2vec representation[3]

For the NN on top of the wav2vec, it was important to stay as simple as possible. So a first idea was to make a classification on every possible command ([full_command_list.txt]) with a convolutional NN. This had already be done for recognizing single words from recordings like in [19] for the Google Speech Commands Dataset [14]. This was quickly aborted for 2 reasons. To begin with, the inputs, the recordings, do not have constant duration (contrary to the Google Speech Commands Dataset that only has 1 second recordings). This means the features extracted have varying shapes (dimensions) and this is not possible to handle without a Recurrent Neural Network (RNN). Of course, it is possible to use zero-padding to force recordings to have the same length, but the difference in duration can be up to 2 seconds, so 32k zeros to had, which does not seem ideal. The second reason was simply that the recordings are not different enough from each other and the NN had trouble differentiate "put red left of blue" from "put red rigth of blue" for example.

III.3.b) Simple Encoder/Decoder 2nd approach

To solve the first issue, I had to use a RNN as discussed before. To solve the second one, I had to make the model predict the command word by word, so rather than making a classification on every possible command, make 4 classifications on the 15 possible words of the vocabulary.

With that in mind, it made sense to use an encoder decoder architecture witch is composed of 2 RNN and can take inputs with varying sizes and can outputs multiple predictions, one for each word. I will not discuss in detail how an encoder decoder works but essentially, it can make multiple predictions with multiple inputs.

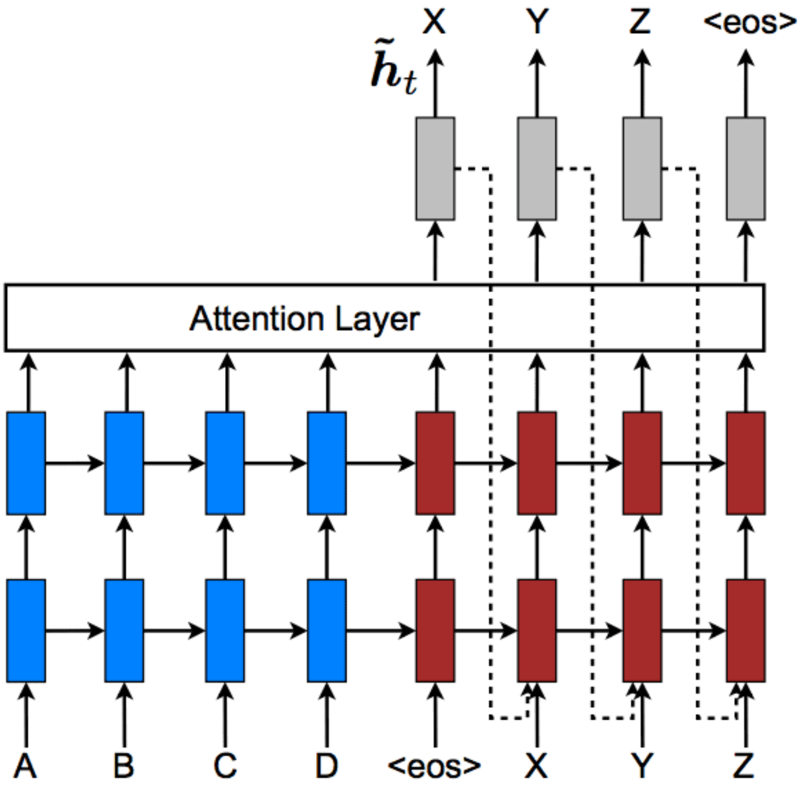


Figure 4: Encoder Decoder Diagram [20]

I will not explain how an Encoder Decoder works but for my problem, A, B, C, D represent the features extracted from the audio and X, Y, Z represent the words predicted.

After implementing the encoder decoder, I trained several models with this architecture [hparams_wav2vec.yaml]. I was not able to pass 40% of accuracy (the accuracy calculation will be discussed in the next part) witch means that the model makes wrong prediction on 60% of the words. It is not random prediction that would have 1/17 ≈ 6% of accuracy but it is not very good. The loss evolution of this model is in annex []

Finally, I tried to switch the wav2vec feature extraction for a classical MFCC feature extraction [hparams.yaml]. This change improved the model greatly so I decided to focus only on this architecture. It seems that the wav2vec was not fitting for this problem so I decided to abandon this approach. Only the results of this final architecture will discussed in the next part (MFCC feature extraction + Encoder Decoder).

III.4 Training and results analysis

III.4.a) Training parameters and metrics

They are several parameters that can be tweaked to improve the model but increase the training time. All of this variables can be modified directly at the beginning of the hparams.yaml file.

- k_fold : Between 1 and 5. 1 means no cross validation and 5 means validation on the whole dataset. This multiples the training time by the number of k_fold.

- apply_data_augmentation : True or False. Appling data augmentation greatly improves the model robustness but also greatly increases the training time

- epochs : number of time the NN is trained on the dataset.

- batch_size : number of data processed in a single batch. Increasing this value lowers the training time but can overload the GPU.

The model was trained using Negative Log Likelihood (NLL) loss. At every epoch, the accuracy is calculated and stored in the log file. The accuracy is the percentage of words the NN predicted correctly.

III.4.b) Training on Google Colab GPU

The benefit of using NN is that once the model is trained, it only consists of matrix of number and do not take a lot of space to store. Also, loading a trained NN and using it to make prediction does not take a lot of computing power. The real downside of working with NN is that it takes a lot of computing power to train them. Generally, NN are trained on the GPU because they are made to perform a lot of simple operation in parallel. A robot generally is not equipped with a good Graphical Card so it makes sense to out-source the training to a powerful computer equipped with a graphic card made for machine learning.

I decided to use Google Colab GPU because it allowed me to use a Tesla T4 graphical card to train the model. This graphical card is much more powerful than the one that I have on my computer and in the Pepper robot. Moreover, this service is free and renting a server equipped with a graphical card is usually very expensive. To do so, I wrote a Google Colab notebook that can be found in the Github repository.

Another benefit of using Google Colab is that anyone can run the notebook and see the training process without having to install anything locally. I put instructions in the Read file in the Github repository.

III.4.c) Results

I will mainly focus on the results of the models trained with 5 folds cross validation. Results for 3 and 1 fold cross validation can be found in annex.

These graphs show the performances of the model trained with and without data augmentation.

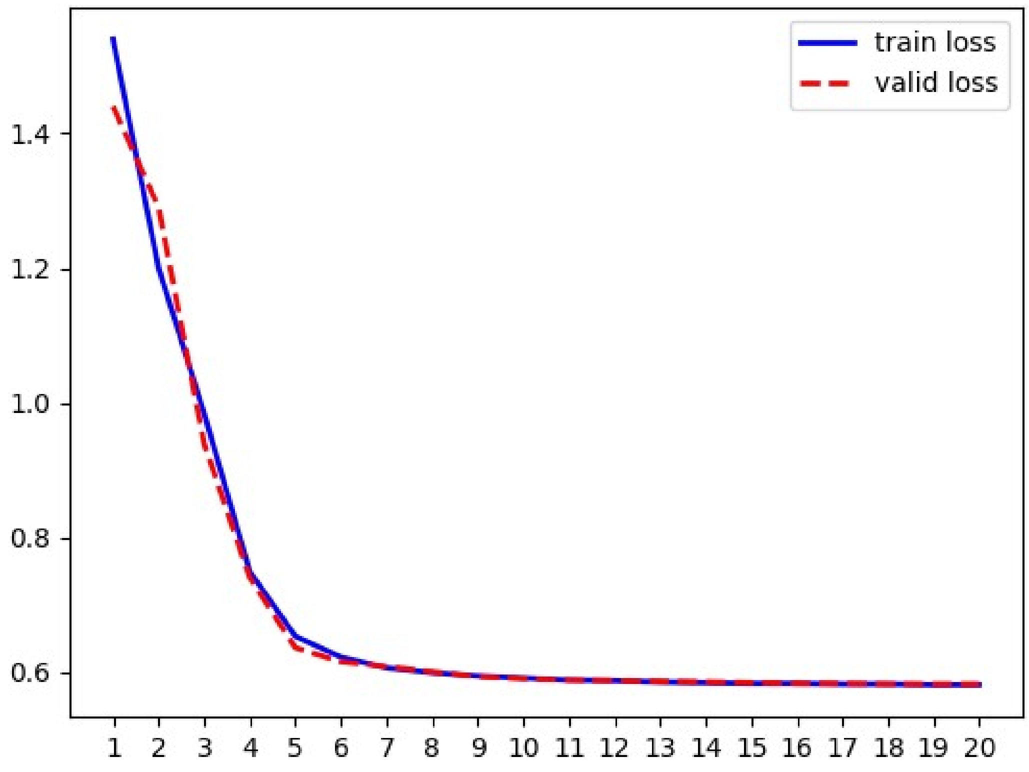


Figure 5: Loss evolution without data augmentation and with 5 fold cross validation

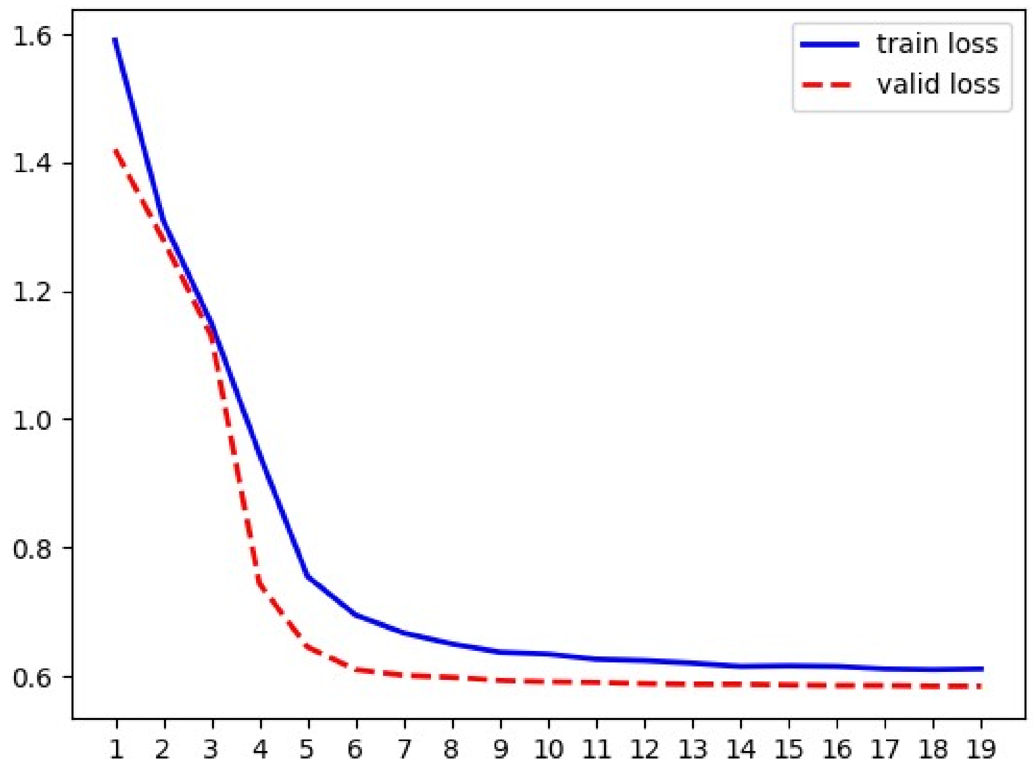


Figure 6: Loss evolution with data augmentation and with 5 fold cross validation

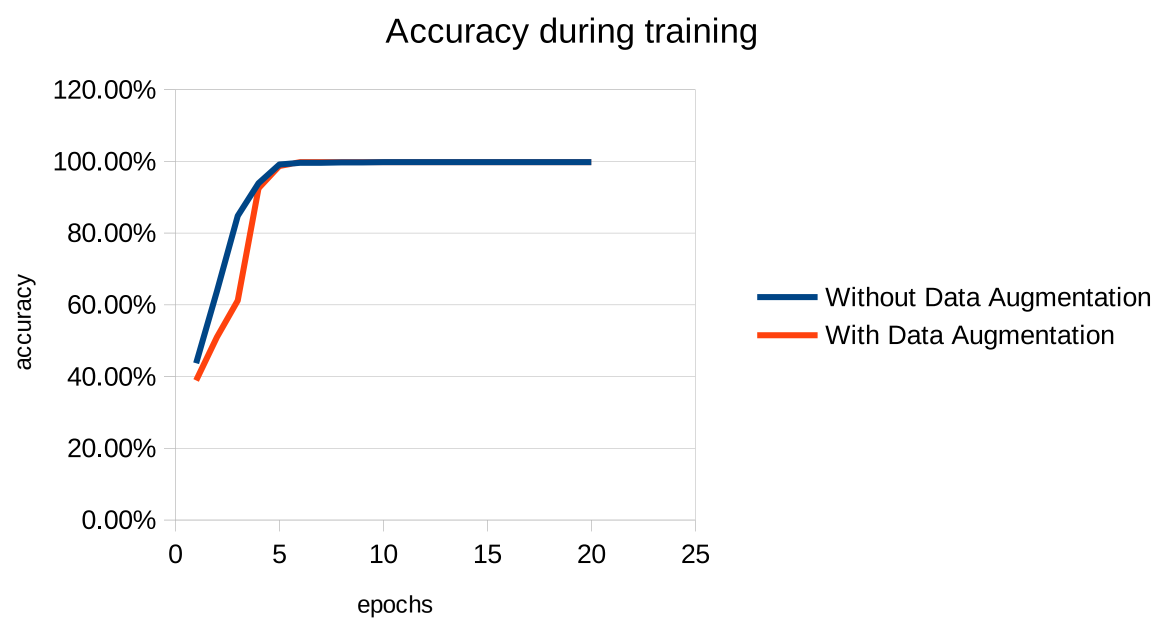


Figure 7: Accuracy evolution with 5 fold cross validation

From this figures, we can see that using data augmentation to train the model makes the training process more difficult at first, but in both cases the NN reaches almost 100% accuracy in about 8 epochs.

The big difference between the two is the training time. In Google Collab GPU, going through 20 epochs took 14 min 40s without data augmentation and 17 min 18 s with. However, not using data augmentation is not possible in term of robustness and flexibility as we will see in IV.1.a.

Using less than 5 fold cross validation does not make sense. Even though having less fold means having a faster training speed per epoch, the NN reaches 100% accuracy faster with 5 folds and is ultimately more performant.

IV. Command Prediction and improvements

IV.1 Model utilisation

IV.1.a) Predictions and robustness in different environments

After I had train the two models (with and without data augmentation) I wanted to see if the model was really able to predict commands, or if it had just *overfitted*. In machine learning, a model may seem very performant because it has a low loss and an almost perfect accuracy, but in reality, the model may just learn *by heart* every label of the data. This means that if the model is presented with a new data entry that is not in the data set, it will fail to recognize it, even though it showed almost perfect accuracy in the training. This is called *overfitting* and is very common when working with a small dataset.

To test this, I wrote a script [inference.py] to load a trained model and use it to make predictions. This is the purpose of a model after all. I also wanted to test the robustness of the two models when confronted with different environment.

Changing the environment means changing the speaker voice and pronunciation, the room to record in, the noise surrounding the user …. I choose 7 different environments to test both models and recorded 10 entries of every environments. I also included the average score that the model gives to each prediction. The score represents how sure a model is when it makes a prediction. A score of 1.0 means that the model is 100% sure of its prediction.

NB: - Natural speaking was represented by me saying "Can you put the red on the blue please ?" instead of "put red on blue".

- Miss pronunciation was simulated by I purposely making pronunciation mistakes and stuttering or hesitating when enunciating the command.

- For the noise, I just played a music while recording and gradually increasing the model for quiet, medium and loud.

Environment	Without Data Augmentation		With Data Augmentation	
	Correct/10	Avg Score	Correct/10	Avg Score
Quiet background noise	10	0.6	10	0.76
Medium background noise	6	0.47	8	0.73
Loud background noise	0	0.33	9	0.73
Natural speaking	2	0.54	8	0.67
Miss pronunciation	2	0.47	9	0.73
Other man voice	6	0.54	10	0.71
Female voice	0	0.31	2	0.52

This table really emphasises the importance of data augmentation with speed variation and noise injection. Once the model without data augmentation was confronted with recording different from the dataset, it was totally lost. On the contrary, the second model was surprisingly robust for every environment change expect for the female voice but that was very expected since it had never heard a high pitch voice. This could be solved in the future by adding pitch variation data augmentation.

IV.1.b) Prediction time

Finally, since the model is meant to work in real-time, it is important that the prediction are instantaneous. The model takes about 0.25 s to make a prediction which is totally satisfying, but this was expected with a small model like this.

However, it takes about 4 s to load the model, but this step must only but ran once at the start of the prediction session.

IV.2 More scalable dataset recording

IV.2.a) More optimized recording process

Even though the dataset is small, it still seems unrealistic to ask users to record that much commands over and over, especially if there are more types of command or more objects in the corpus, the number of recording would increase exponentially. A big problem is that there is an uneven distribution of words in the data set.

Uneven distribution of words not only means that there are unnecessary recordings, but it may also lead to the model is being bias towards more common words in the dataset.

To solve that, I wrote a script that takes a minimum number of utterance each word must recorded, and outputs only the minimum of command to record to satisfies that requirement. This evens out the distribution and lowers the number of recordings to make:

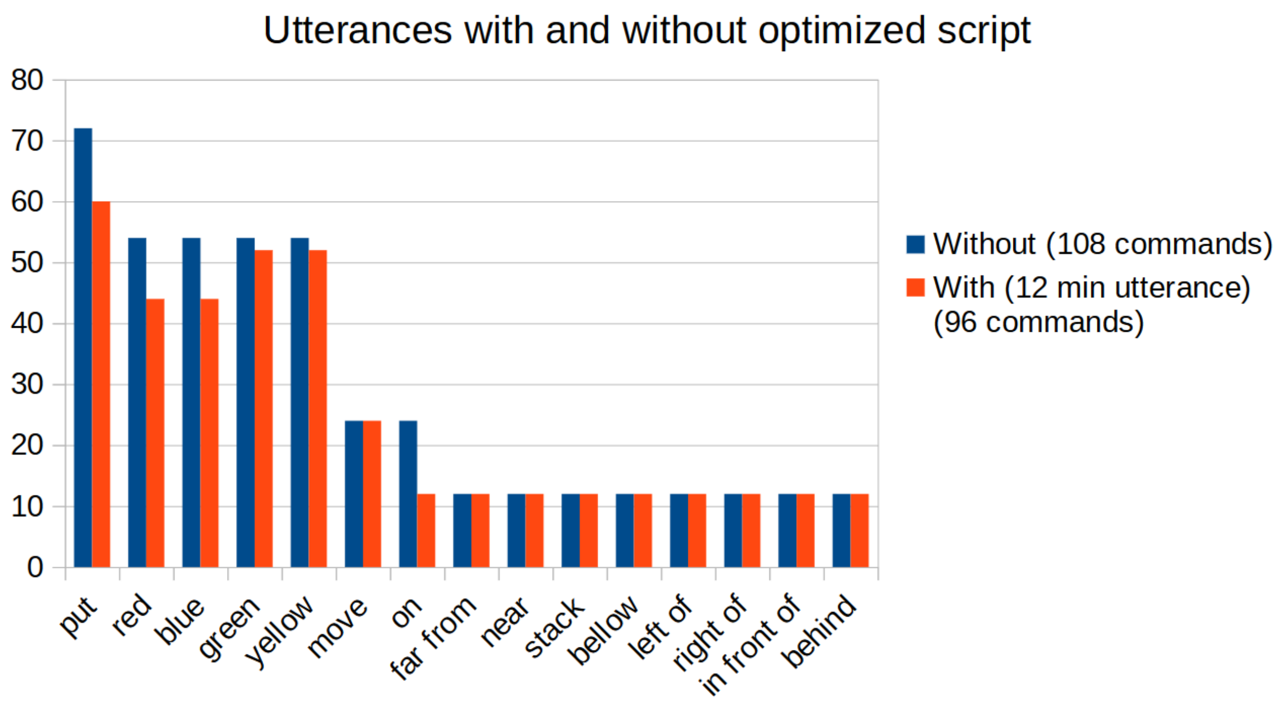


Figure 8: More even distribution with optimized command choice

This method will only get more efficient with the number of commands, object or preposition increasing, because words that are already recorded sufficiently will not have to be recorded again.

For example if we had “orange” to the objects :

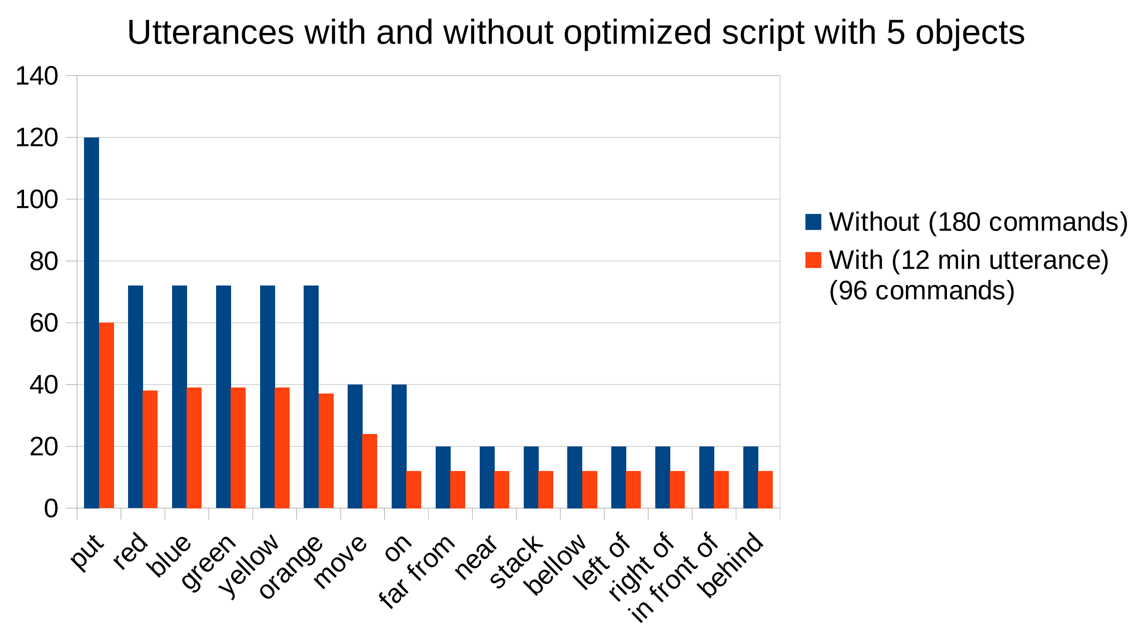


Figure 9: Adding another object does not increase the number of commands

IV.2.b) Creating new data without new recording

Since the commands are very similar, an idea was to create data by cutting parts of recordings and assembling them. For example, if the user records “put blue on red” and “stack yellow on green” , we could cut out “put blue” , “on red” , “stack yellow” , “on green” and assemble “put blue on green” and “stack yellow on red” without having to record them. Nevertheless, the resulting audio would not be near as good as a real recording because there would be a noticeable cut in the middle. This problem gets worse if we cut records word by word. However the number of data created would be very significant.

I started implementing this idea but I did not have time to finish it because it implies a lot anticipation before calculating all the possible recording to assemble. Indeed, if the commands are cut word by word, the number of possible recordings to assemble gets really high and it takes too long to process.

NB: It is not very realistic to ask the user to record the words separately and create the dataset by assembling the word recording. This was tested by Dr. Cuayahuitl and did not work. The main reason is that we pronounce words differently whether they are alone or placed in a sentence. Although assembling audio from full commands seems to be a very promising improvement.

V. Personal impressions and what remains to be done

This internship was really instructive regarding speech recognition and NN. The fact that I had to do everything by myself helped me understand all the components that goes in making a Speech Recognition model, from making the dataset, to training the NN, to using it to make predictions. I am now more familiar with RNN and more complex types of NN. I gained valuable experience on the SpeechBrain library (and on Pytorch) which seems very promising and will likely become the reference framework for developing speech related NN.

I also learned to manage a project by myself and to organize the different steps with the limited time I had. I needed to come up with new ways of approaching speech recognition. This was really interesting because there was no real work that had been done of the same kind. I could not copy some code of the Internet, I wrote almost every line of code myself except for the part that were required by SpeechBrain. As a result, I learned of to use Github witch is essential for code oriented project, even when not working as a team.

A lot remains to be done to think about using this approach. First of all, the recording time of the user must be decreased a lot to even consider asking it to record the dataset. Also, the model was never tested on the Pepper Robot, the integration in itself can be a whole new project. The vocabulary must be increased and it would be preferable if the user could user natural speaking, and even better, spontaneous speaking, but that seems impossible with so few data.

Nevertheless this work can be interesting if one wanted to avoid using big dataset and big GPUs, avoid using the GAFAM speech recognizers and wanted to build its own speech recognizer from scratch. Not many work has been done in this direction and it could be something to explore in the future.

VI. Bibliography

[1]..........A. Raju, D. Filimonov, G. Tiwari, G. Lan, et A. Rastrow, « Scalable Multi Corpora Neural Language Models for ASR », *ArXiv190701677 Cs*, juill. 2019, Consulté le: août 01, 2021. [En ligne]. Disponible sur: http://arxiv.org/abs/1907.01677
[2]..C.-C. Chiu *et al.*, « State-of-the-Art Speech Recognition with Sequence-to-Sequence Models », in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Calgary, AB, avr. 2018, p. 4774-4778. doi: 10.1109/ICASSP.2018.8462105.
[3]A. Baevski, H. Zhou, A. Mohamed, et M. Auli, « wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations », *ArXiv200611477 Cs Eess*, oct. 2020, Consulté le: juill. 26, 2021. [En ligne]. Disponible sur: http://arxiv.org/abs/2006.11477
[4]........J. Garofolo *et al.*, « TIMIT Acoustic-phonetic Continuous Speech Corpus », *Linguist. Data Consort.*, 1992.
[5]........V. Panayotov, G. Chen, D. Povey, et S. Khudanpur, « Librispeech: An ASR corpus based on public domain audio books », in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, avr. 2015, p. 5206-5210. doi: 10.1109/ICASSP.2015.7178964.
[6].....S. Lathuilière, B. Massé, P. Mesejo, et R. Horaud, « Deep Reinforcement Learning for Audio-Visual Gaze Control », 2018.
[7]..I. Moreira, J. Rivas, F. Cruz, R. Dazeley, A. Ayala, et B. J. T. Fernandes, « Deep Reinforcement Learning with Interactive Feedback in a Human-Robot Environment », *CoRR*, vol. abs/2007.03363, 2020.
[8]....M.-A. Zamani, S. Magg, C. Weber, S. Wermter, et D. Fu, « Deep reinforcement learning using compositional representations for performing instructions », *Paladyn J Behav Robot.*, vol. 9, nº 1, 2018.
[9]....E. Bastianelli, G. Castellucci, D. Croce, R. Basili, et D. Nardi, « Effective and Robust Natural Language Understanding for Human-Robot Interaction », in *Prestigious Applications of Intelligent Systems (PAIS)*, 2014, vol. 263.
[10]...Y. Tada, Y. Hagiwara, H. Tanaka, et T. Taniguchi, « Robust Understanding of Robot-Directed Speech Commands Using Sequence to Sequence With Noise Injection », *Front. Robot. AI*, vol. 6, 2019.
[11]..M. Ravanelli *et al.*, « SpeechBrain: A General-Purpose Speech Toolkit », *ArXiv210604624 Cs Eess*, juin 2021, Consulté le: juill. 13, 2021. [En ligne]. Disponible sur: http://arxiv.org/abs/2106.04624
[12]....U. India, « Tensorflow or PyTorch : The force is strong with which one? », *Medium*, nov. 14, 2018. https://medium.com/@UdacityINDIA/tensorflow-or-pytorch-the-force-is-strong-with-which-one-68226bb7dab4 (consulté le juill. 19, 2021).
[13]...R. Scalise, S. Li, H. Admoni, S. Rosenthal, et S. S. Srinivasa, « Natural language instructions for human–robot collaborative manipulation », *Int. J. Robot. Res.*, vol. 37, nº 6, p. 558-565, mai 2018, doi: 10.1177/0278364918760992.
[14].......P. Warden, « Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition », *ArXiv180403209 Cs*, avr. 2018, Consulté le: juill. 27, 2021. [En ligne]. Disponible sur: http://arxiv.org/abs/1804.03209
[15].........J. Ramı́rez, J. C. Segura, C. Benı́tez, Á. de la Torre, et A. Rubio, « Efficient voice activity detection algorithms using long-term speech information », *Speech Commun.*, vol. 42, nº 3, p. 271-287, avr. 2004, doi: 10.1016/j.specom.2003.10.002.
[16].............................« WebRTC », *WebRTC*. https://webrtc.org/?hl=fr (consulté le juill. 30, 2021).

[17]…J. F. Ramirez Rochac, N. Zhang, J. Xiong, J. Zhong, et T. Oladunni, « Data Augmentation for Mixed Spectral Signatures Coupled with Convolutional Neural Networks », in *2019 9th International Conference on Information Science and Technology (ICIST)*, 2019, p. 402-407. doi: 10.1109/ICIST.2019.8836868.
[18] T. Ko, V. Peddinti, D. Povey, M. L. Seltzer, et S. Khudanpur, « A study on data augmentation of reverberant speech for robust speech recognition », in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, mars 2017, p. 5220-5224. doi: 10.1109/ICASSP.2017.7953152.
[19].........S. Majumdar et B. Ginsburg, « MatchboxNet: 1D Time-Channel Separable Convolutional Neural Network Architecture for Speech Commands Recognition », *ArXiv200408531 Eess*, avr. 2020, Consulté le: juill. 27, 2021. [En ligne]. Disponible sur: http://arxiv.org/abs/2004.08531
[20]….M.-T. Luong, H. Pham, et C. D. Manning, « Effective Approaches to Attention-based Neural Machine Translation », *ArXiv150804025 Cs*, sept. 2015, Consulté le: juill. 30, 2021. [En ligne]. Disponible sur: http://arxiv.org/abs/1508.04025

VII. Annex

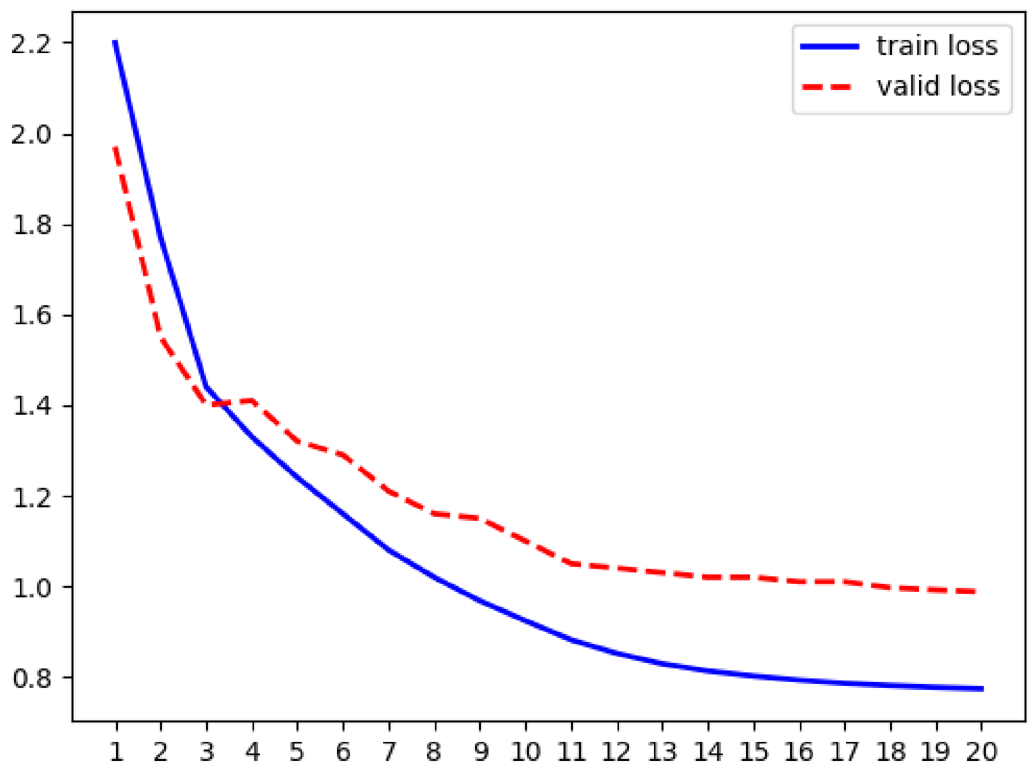


Figure 10: Loss evolution with 1 fold without Data Augmentation

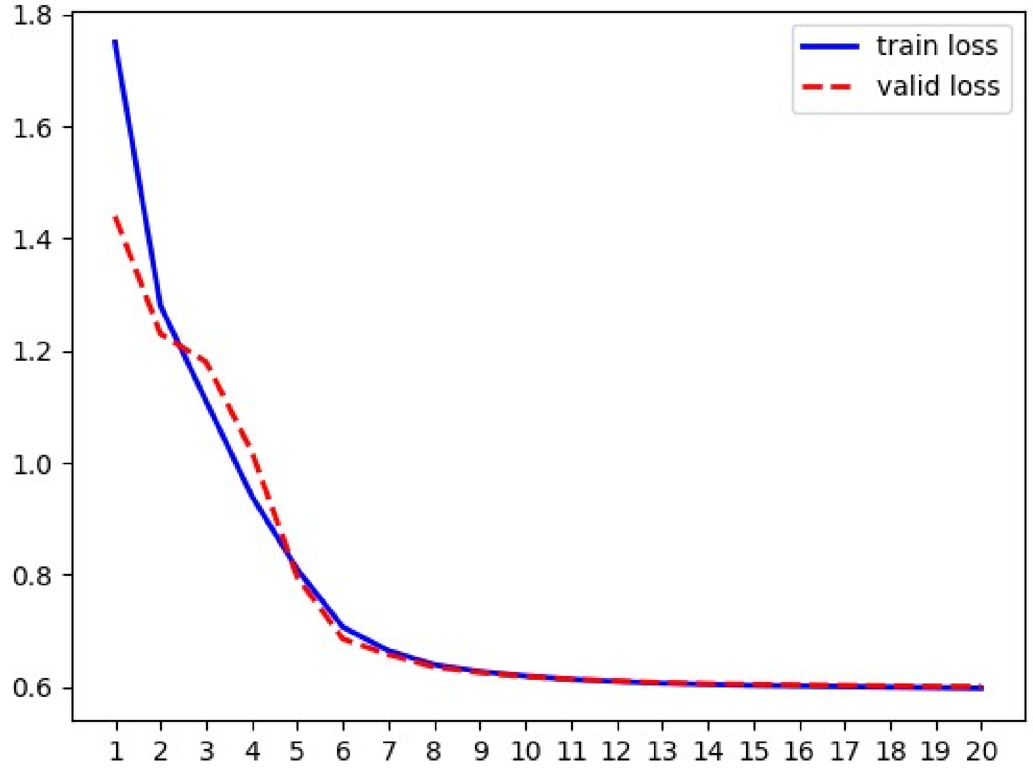


Figure 11: Loss evolution with 3 fold without Data Augmentation

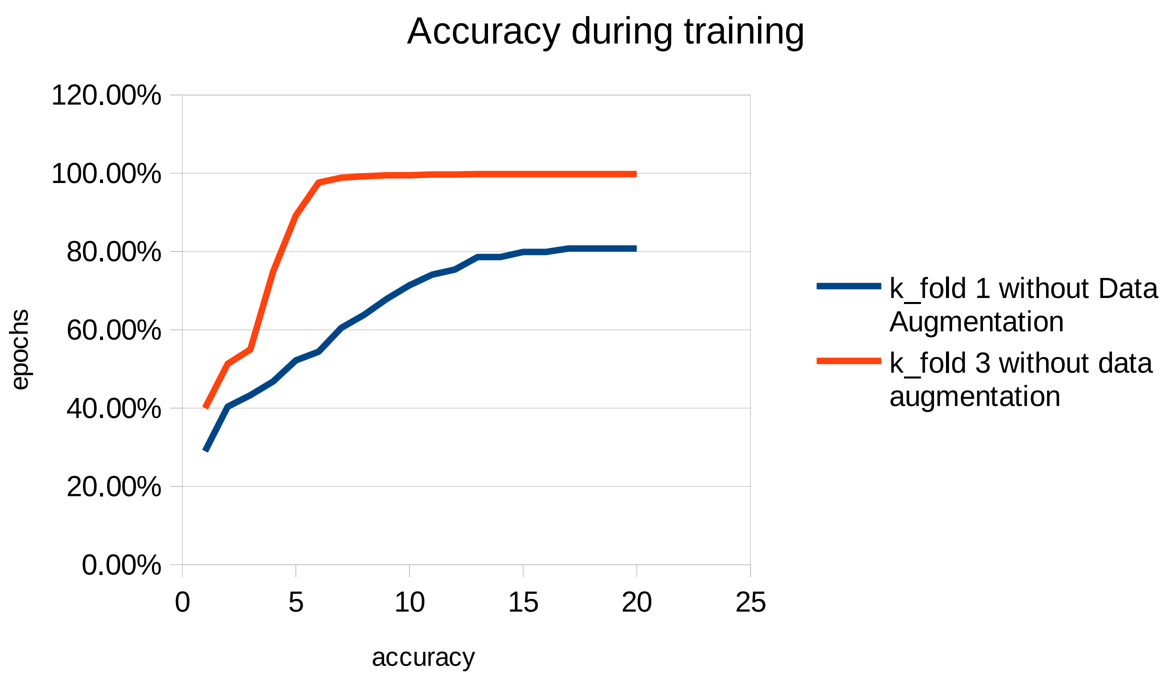


Figure 12: Accuracy evolution with 3 and 1 fold cross validation