Speech-To-Text Transcription Using Neural Networks

Training of a Spanish STT model using the DeepSpeech engine

Daniel Finca Martínez

Universidad de Oviedo
Universidá d'Uviéu
University of Oviedo

Tutor: Maria Esperanza García Gonzalo
Cotutor(s): Juan Díaz Suárez,
Jose Ignacio de la Vega Sánchez

Resumen

Este proyecto nace de la necesidad de reemplazar a servicios de transcripción en la nube por una solución propia para una empresa de servicios informáticos. A través de múltiples herramientas y técnicas de Machine Learning, se plantea crear un modelo que permita la transcripción de audios de voz en español a texto.

Una de estas herramientas es el proyecto de código abierto llamado DeepSpeech, desarrollado por Mozilla. Dicho proyecto, permite a sus usuarios entrenar modelos de inteligencia artificial que transforman audio de cualquier idioma en texto, lo que se conoce como Speech-To-Text en la literatura.

Gracias a un proceso compuesto de varias fases se consigue crear un modelo que deja de manifiesto la capacidad de operación de estos sistemas y su potencial para emular a servicios de pago existentes en el mercado actual.

Tras una breve introducción teórica al problema Speech-To-Text, este documento explica con detalle todas las fases de dicho proceso: recolección de datos, preprocesamiento, preparación del entrenamiento, entrenamiento, obtención de resultados y evaluación de los mismos. Además, se incluyen también un serie de ampliaciones posibles a todo este proceso que añaden valor y funcionalidad al producto final.

Abstract

This project arose from the need to replace cloud-based transcription services with a proprietary solution. Through multiple Machine Learning tools and techniques, the idea is to create a model that allows the transcription of audio speech in Spanish into text.

One of these tools is the open-source project called DeepSpeech, developed by Mozilla. This project allows its users to train Artificial Intelligence models that transform audio from any language into text, known as Speech-To-Text in the literature.

By means of a process consisting of several phases, a trained model is created to demonstrate the operational capacity of these systems and their potential to emulate existing paid services on the current market.

After a theoretical introduction, this document explains in detail all the phases of this process: data collection, preprocessing, training preparation, training, results collection, and evaluation. In addition, the document includes a series of possible extensions to this process that add value and functionality to the final product.

Contents

List of Figures

List of Tables

Listings

1 Introduction

1.1 Motivation

Many would say that technology had its limits some years ago, yet now we have mini-computers with us all the time in the form of mobile phones. We can also buy things with internet money or pay with our phones or smartwatches. The world of technology is in constant evolution.

In the fields of Machine Learning, Big Data, Human-Computer interaction, there are countless applications to solve everyday tasks. That sometimes would defy common sense and even make us question our purpose in our daily jobs, knowing that someday, machines could replace us completely. Fortunately for human beings, there are still lots of things machines can not do (yet).

In particular, Natural Language Processing (NLP) has made huge improvements over the years with the development of newer Deep Learning techniques as well as higher availability of data and computing power.

This work aims to contribute to this NLP world by leveraging existing technologies to create a tool that transcribes speech into readable words of the Spanish language.

This project was initially proposed by a company centred around the world of communication, management and establishing better relationships among people. This company is named Alisys and their objective with this project was to improve their current speech-transcribing system and hopefully create a product of their own, without relying on the heavy players that dominate the industry nowadays (Google or Amazon, for instance). They did not plan to improve existing services, just to emulate them close enough for them not to depend on other companies or services.

1.2 Speech Recognition Systems

Around the 1940s, the first computer prototypes were under construction [6]. However, these were not that powerful compared to what we have nowadays. This fact did not stop scientists and engineers from asking questions and seeking progress.

One of the many questions they sure asked themselves was: Could we possibly mimic human behaviour with one of these machines? And sure they could, but it was a complex and ambitious task. We know for sure they were not wrong. We have made many approximations to mimic human behaviour.

These approximations include the processing of the human language. We are talking about Speech Recognition Systems (SRSs). The terms Speech Recognition (SR) and Automatic Speech Recognition (ASR) are synonyms in the literature. The latter is the most common one. This concept represents an entire subfield of Computer Science along with Computational Linguistics. It aggregates contributions from many disciplines.

They aim to study, design, and implement SRSs that process human speech. The objective is to make a computer "understand" and "interpret" what a human or a group of them are saying, reacting to these stimuli in a wide range of possibilities. The basic idea is to convert acoustic signals to words [19].

The basis of these systems is training, a process through which SRSs learn what to look for when operating. The quality of the data used as input will determine how well the system behaves when processing new data. During this process, the system receives human speech as input. It is analyzed to fine-tune its perception of how human speech sounds. The greater the amount of data provided, the better the system will be when generalizing new input speech, although this is not true in every scenario.

From the technological perspective, ASR has a long history that includes many major innovations. One of these is the explosion of Big Data and Deep Learning. These two massive interdisciplinary fields have given ASR a lot of room to expand and improve over the last two decades, even during the late 1990s [3].

1.2.1 Some history about the evolution of SRSs

The first mechanical device that produced artificial speech was presented to the public as far back as in the early 1780s. Its creator, Mr Wolfgang von Kempelen, made it public during a journey through Europe where he showed yet another of his inventions called "Mechanical chess player". On top of that, Sir Thomas Alva Edison invented the first dictation machine around 1879 [2, 14]

Later on, between the decades of the 1950s and 1970s, this invention was followed by "Audrey" (created by Bell laboratories in 1952), a system that could recognize a single voice reading digits out loud. Ten years later, IBM showed its creation called "Shoebox". This machine could understand and answer a total of 16 English words [24, 13].

After these great contributions, during the 1970s and 1980s, the ASR field saw great advancements. These include the system "Harpy" devised in the university of Carnegie-Mellon around 1976 capable of understanding a thousand plus words [25]. However, it had some limitations as speakers would need to slow down their pace for the machine to understand them. During the 1980s, IBM Tangora, described as the world's fastest typist, was made public. This machine could predict upcoming phonemes in the user's speech. It was based on Hidden Markov Models (HMM) [26]. However, as with many of these revolutionary inventions, it had problems with fast and unclear speech and background noise [17].

During the 1990s, as the internal machinery of computers got better and faster, several SRSs sprouted, such as Dragon Dictate and BellSouth's voice portal dial-in interactive voice recognition system (VAL) [24].

The decades of the 2000s and 2010s saw great improvement over the years. First, the National Security Agency (NSA) started using speech recognition in 2006, specifically isolated word recognition [10]. Later on, in 2008, Google launched the very first voice search app, making voice and speech recognition accessible through mobile devices [23]. Besides, Apple released Siri in 2011 presented as the new portable digital assistant integrated into the iPhone 4S [8].

Since 2012, the explosion of Deep Learning displaced the usage of HMM-focused models to include newer approaches such as End-to-End models, which came in proposing an all-in-one processing strategy: data came as input into the model, in this case a neural network, and the desired output came out. This procedure has one element susceptible to optimization. Whereas, in the traditional speech-processing approach, we would need to optimize several separate steps [22].

1.2.2 Currently existing SRSs

At present, the IT market has several products that implement ASR technologies. These products provide services spread all around and are present in our daily lives. Many companies offer these services, either as a paid subscription or free-to-use embedded in other products such as desktop computers, laptops, smartphones and domotics.

Some of these below-mentioned products have been developed by the largest tech companies in the world today. These companies include Apple, Microsoft, Alphabet (Google) and Amazon (Largest Companies by Market Cap). Below we find a breakdown of some products classified by owner company:

- Apple
  - Siri Personal Assistant
  - Mac device Voice Control and Apple Dictation
- Microsoft
  - Windows Speech Recognition 1 and Windows Speech Recognition 2
  - Azure Speech-to-Text
  - Azure Text-to-Speech
  - Azure Custom Speech
- Google
  - Google Cloud Speech-to-Text
  - Google Dialogflow
  - Google Assistant
- Amazon
  - Alexa
  - Amazon Transcribe

1.2.3 Types of SRS

The human voice is a complex thing to process computer-wise. Speech Recognition aims to transcribe speech (STT) or synthesize it (TTS). It should not be mistaken for the same thing that is Voice Recognition. The former only processes the speech as potential textual information or vice versa. The latter worries about identifying the speaker by the sound of their voice.

The most common SRSs fall under one of the following categories. With regards to the objective of the system, we have:

- **Speech-To-Text (STT):** This approach aims to transform spoken words, commonly known as speech, into readable text later processed by a program. Such as creating intent-based conversational bots or virtual assistants, for example.
- **Text-To-Speech (TTS):** The goal of this other approach is the opposite of the first one. It aims to transform plain text into human-like speech. Comparatively, it is harder to do this if we want to obtain a reliable system that sounds like a human voice. Possible applications could be virtual assistants or voice emulation programs.

With regards to the context the system generates for the input speech [16]:

- **Isolated Word Recognition:** This approach processes each word separately without taking any other input into context. Applications of this approach could be command-based applications.
- **Continuous Speech Recognition:** This approach takes longer segments of speech as input instead of separated words. One possible drawback to this one is that it takes prior and following input as context to the input word. Usage of this approach would involve conversational systems like chatbots.

Serve as a side-note, every piece of speech is affected by many factors, often not under control. Some of these could be background noise, the speaker's emotional state, or voice quality. Every one of these things can affect how the system processes input speech. The probability that these factors manifest in continuous speech is much higher than that of the isolated words approach.

1.2.4 Possible applications

Nowadays, we humans use our voices as the basis for our communication. Among other features, the voice is something unique to one another. It is part of our biometrical data. Many programs and everyday applications use this information to provide services to us like personal assistants, on-the-fly note-taking, and many others. These are some of the most cutting-edge applications of these systems [19]:

- Automatic device control
- Telephone operator's jobs automation
- Program data entry
- Program interface voice control
- Creating input data to Natural Language Processing systems

1.2.5 What they represent in the context of the project

This work leverages Artificial Intelligence to create a system that can transcribe audio samples into readable text. These ASR models have introduced a new and revolutionary era for task automation.

We have created machines that do simple things for us. The more computers developed, the harder those "simple things" could get. Such tasks are growing more complex as we involve human end-users by processing speech. Years ago, we would not expect a machine to "understand" words, but now it seems ordinary, and often we miss having our phone close to read some book for us or to give us the weather forecast.

Thanks to many advanced techniques, we can now rely on computers to perform these complex human-machine interactions to focus on other things. These interactions hence get minimized. That allows us to optimize the time and resources needed to perform a task and spend them elsewhere.

With these two latter resources being of the essence, I also want to stress a saying that got to me: "If you spend 5 minutes doing something by hand, but you would take 5 hours failing to automate it, then you should go for the second option."

On the one hand, it is clear nowadays that machines are better than us at performing repetitive and relatively simple tasks.

On the other hand, machines are not better than humans at performing tasks that require complex decision-making or intricate human interaction.

1.3 Recurrent Neural Networks in ASR

The main purpose of this kind of Neural Network is to process sequential data. Such data is processed one slice at a time. Naturally, this provides less knowledge than processing the whole sequence and the context around these slices.

Natural Language Processing uses these networks to solve problems such as predictive writing. The system needs to consider previously written words to "guess" what the user can type next since words inside sentences are not independent.

This problem is hard to solve using traditional neural network models because of their topology. These are usually composed of input and output layers and optionally 1-n hidden layers. The neurons on these layers are often fully connected to their neighbour layers.

Neurons of a single layer in a non-RNN topology are not connected. This does happen in RNNs. You can find neurons connected in a single layer of the network. This allows inputs and outputs from such layer's neurons to affect each other and thus the final output of the recurrent layer.

The series of outputs obtained from a sequence or data slice depends on the previously calculated output. This is what gives the name to RNNs. Some of these networks can store previously generated data to use in further data slices of the current input sequence. The thing is that not every network uses these devices (usually referred to as "memory" in the literature). In this specific case, DeepSpeech architecture does not use any memory cells inside their network [12, 4].

1.3.1 Basic elements: Recurrence and Memory in RNNs

To explain the concept of memory and recurrence, we will need to set up a small mathematical notation. Let us define a T length input sequence as X, where $X = \{x_1, x_2, ..., x_T\}$, such that $x_t \in \mathbb{R}^N$ is an input vector at time t. Now, we define the so-called "memory" that includes up to t times (also included) as h_t. Then, we can define the output o_t as:

$$o_t = f(x_t, h_{t-1})$$

where the function f maps memory and input to a single output. Memory data comes from the time instant $t-1$ while input x_t is from the current instant. For the initial case x_1, as there is no previous step, the memory value h_0 is the null vector 0.

If we then consider that these series of outputs o_t summarize the entire history of memories h_{t-1} and inputs x_t up to, and including t, we obtain the following equation:

$$h_t = o_t = f(x_t, h_{t-1})$$

Hence, we define the term "recurrence" as the application of the same function for each time instance t, where the output is directly dependent on the previous result of the function.

To apply this concept to neural networks, we need to include a couple of key elements to these structures:

$$h_t = f(Ux_t + Wh_{t-1})$$

where W and U are weight matrices $W, U \in \mathbb{R}^{(NxN)}$, and f is a non-linear function, such as tanh, σ, or ReLU. The next figure (1) shows the structure of a series of recurrent neurons applying these concepts [15].

The objective of this explanation is to resemble the concept of memory and recurrence but done through equations whose outputs depend directly on all the previous outputs of the same equation for a given sequence.

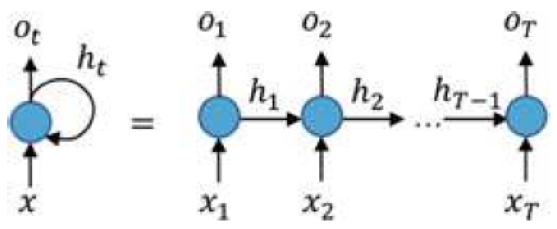


Figure 1: Example of a recurrent neural network

1.3.2 Vanishing gradient problem and regularization

A consequence of the training of RNNs is that the backpropagation algorithm often generates an effect that is commonly known as the Vanishing Gradient Problem or Exploding Gradient Problem. Now, this happens due to the recurrence existing in the network and the operations performed in it.

In essence, training an AI model involves many matrix multiplications. The operations performed by RNNs include these as well. These recurrent operations make the gradients propagated through the network grow or shrink exponentially. So with any major or minor irregularity in the data, the gradients will "explode" or "vanish" according to the error calculated at each time step. The recurrent multiplication in the backpropagation step causes an exponential effect for any irregularity. That is,

- If the weights are small, the gradients will shrink exponentially.
- If the weights are large, the gradients will grow exponentially.

Another relevant case scenario is when the contribution to the error is so small that the weight update is mostly negligible and thus may lead the network to stop training.

One way to soften the impact of this effect is to initialize the weights of the network in a controlled way. However, even with careful initialization, it can still be challenging to deal with long-range dependencies. Zero could be a common initialization for RNNs for the hidden state. Moreover, the performance can typically be improved by learning this hidden state.

There are many ways to combat this problem: for instance, focusing on careful initialization or controlling the size of the gradient being propagated. The way most commonly used to combat vanishing gradients is the addition of gates to RNNs. RNN sequences can be very long. For example, if an RNN used for speech recognition samples 20 ms windows with a stride of 10 ms, it will produce an output sequence length of 999 time steps for a 10-s clip (assuming no padding). Thus, the gradients can vanish/explode very easily [15].

Serve as a sidenote, 10 second-long is the maximum length that the DeepSpeech framework allows for input audios during training.

1.3.2.1 Gradient clipping

This is the concept that aims to control the range of the gradient during training. It is used to prevent and limit gradient explosion, forcing it to a specific range. This limitation can solve many problems, specifically preventing overflow errors during training [15].

The two most common ways to clip gradients are:

- L_2 norm clipping with a threshold t.

$$\nabla_{new} = \nabla_{current} \circ \frac{t}{L_2(\nabla)}$$

- Fixed range with a maximum threshold t_{max} and a minimum t_{min}

$$\nabla_{new} = \begin{cases} t_{min} & \text{if } \nabla < t_{min} \\ t_{max} & \text{if } \nabla > t_{max} \end{cases}$$

1.3.2.2 Backpropagation through time (BPTT) sequence length When training RNNs, we have to bear in mind that the computation carried out highly depends on the number of time steps in the input data. We can limit the amount of computation during training by setting a maximum sequence length for the training procedure.

A few common ways to do this:

- To pad all the training data to the longest desired length.
- To truncate the number of steps that are backpropagated to the network during training.

Moreover, during the early steps of training, we can overlap sequences with truncated backpropagation to help the network converge quicker. If we increase this overlapping duration amidst training also helps the network to converge earlier in the training [15].

Setting a maximum sequence length can be useful in a variety of situations. For instance:

- When a static computational graph requires a fixed size input.
- If the model happens to have running memory constraints.
- If gradients are very large at the beginning of the training procedure.

1.3.2.3 Recurrent dropout It is well known that deep learning topologies are prone to overfitting. These models are prepared to receive huge amounts of data for training and are expected to output predictions for newer, unknown data.

The problem is that many times, these networks have a lot of parameters to adjust (biases, weights, ...). These tend to fit the training data and lead to overfitting. Thus, rendering the model useless towards new data since it will have memorized the training and not learned how to predict new results.

Recurrent Neural Networks are not an exception to this fact. Dropout, among other techniques, is a common regularization approach. It is also really intuitive to apply to Recurrent Networks. The thing is that the basic idea of dropout will not suffice. It must be tuned to work on RNNs.

If the original form of dropout is applied at each step, then the combination of masks can cause a small amount of signal to be passed over longer sequences. Instead, we can reuse the same mask at each step to prevent loss of information between time steps [15].

1.3.3 Basic structure of a BRNN

Generally, when predicting new outcomes for the input data, we can only expect to know about input from the previous timesteps. Well, if we knew about both past and future timestamps outputs, we should be able to generate better predictions for a time t

by using more information. This is achieved thanks to Bidirectional Recurrent Neural Networks.

This type of network is run over two opposite sequences: One of these sequences runs in the forward direction, and the other runs in the backward direction. For an input sequence $X = \{x_1, x_2, ..., x_T\}$, our forward context receives the inputs in forward order with regards to the timestamp series $t = \{1, 2, ..., T\}$. However, the backward context receives the inputs in reverse order like so: $t = \{T, T-1, ..., 1\}$. These two structures constitute a single bidirectional layer.

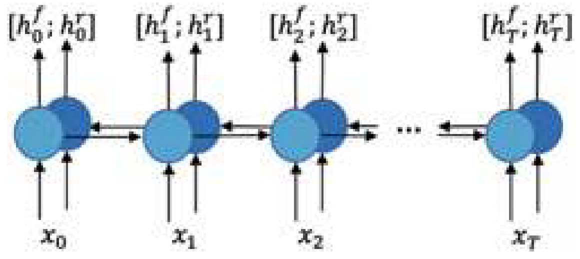


Figure 2: Example of a bidirectional neural network

In figure 2, the outputs are concatenated to form a single output vector $[h_t^f; h_t^r]$ where $t \in \{0, 1, ..., T\}$ holding the forward and backward context. But usually, the output of the two RNNs, h^f and h^r, are often joined to form a single output vector either by summing them, concatenating, averaging or a different method [15].

This type of structure can be used in many NLP scenarios. For example, it has proven useful when classifying phonemes in speech recognition as knowledge on future contexts contributes with better information for newer predictions for a given time slice t.

These networks typically outperform forward-only RNNs in most tasks. Plus, this approach can be extended to other forms of recurrent networks such as bidirectional LSTMs (BiLSTM) [15].

This comes with some limitations as to how it operates. Bidirectional RNNs must know the full input sequence before generating predictions because the reverse RNN requires x_T for the first computation of the sequence. Hence, bidirectional RNNs cannot be used for real-time applications or Just-In-Time translation. However, depending on the requirements of the application, having a fixed buffer for the input can help cope with this constraint.

1.4 End-to-End Speech Recognition

As mentioned in previous sections of this work, the approach followed for speech recognition and audio transcription relied on complex engineered pipelines that processed the data step by step. These splitting techniques generated some errors that would not be propagated to the next steps. This rendered into sub-optimal results leading to model sensitivity towards speaker variation and noise. With some evolution, a transition to more complex structures took place that allowed models to learn directly from the data instead of relying on these complicated pipelines. These end-to-end models aim to optimize the predictions made rather than separating them into several steps.

With end-to-end modelling, the input–target pairs only need to be the speech utterance and the linguistic representation of the transcript. We have many possible representations: phonemes, triphones, characters, character n-grams or words. Among these, we would obviously choose words in ASR as it is the expected result to be extracted from the speech utterance. Though there is a problem; We would need an enormous output layer so that we can represent the whole vocabulary of the language. Plus, we would also need to have training samples for all the words in the vocabulary. Hence, falling into lower accuracies than other possible representations.

In recent times, end-to-end approaches have moved towards using characters, character n-grams, and some word models plus enough training data. These data pairs can be easier to produce since it is not needed to obtain data for each possible word in the target language.

The core component of end-to-end ASR must have a way to replace the Hidden Markov Models (HMM) with something also able to model the temporal structure of speech. The most common methods are "CTC" or "Connectionist Temporal Classification" and "attention" [15].

1.4.1 Connectionist Temporal Classification (CTC)

The usage of the conventional Hidden Markov Model (HMM)-based approach requires data to be pairwise aligned. As in: "This subset of audio features corresponds to this phoneme". Obtaining these alignments can get absurdly expensive for large datasets. Ideally, these alignments would not be needed for any utterance-transcript pair. The CTC method was introduced to help label unsegmented sequences directly. This way, we do not need to split the process into alignment and inference.

Given an acoustic input $X = [x_1, x_2, ..., x_T]$ of audio features as well as the output sequence $Y = [y_1, y_2, ..., y_U]$. There is no certainty of obtaining an accurate alignment of both sequences X and Y. Plus, the lengths of both can vary. Often $T \gg U$, for instance, when there is silence in the input audio.

These alignments could be generated following a simple theoretical correspondence (see figure 3). For each input audio feature x_t, there will be exactly one character prediction. There will be repetitions, but these can later be merged into one. Now, this generates two issues:

1. In the recognized speech, we could find periods of silence that do not necessarily correspond to any output,
2. And also, we do not allow for repeated characters in the output because repetitions are merged.

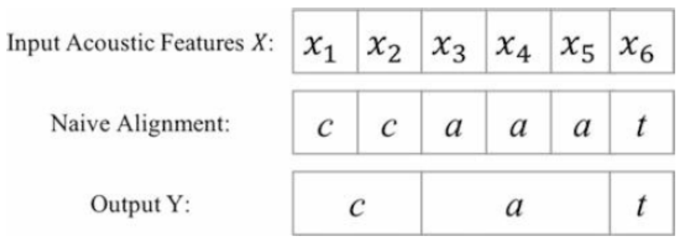

Input Acoustic Features X:
x_1 x_2 x_3 x_4 x_5 x_6
Naive Alignment:
c c a a a t
Output Y:
c a t

Figure 3: A naive approach to an alignment of an input X of length 6 and an output $Y = [c, a, t]$

These issues are solved by the CTC algorithm by introducing a blank token that acts as a null delimiter. Such token is to be removed after the collapse of two-character predictions. That lets repeated sequences remain (such as the word "hello"). Plus, it creates a correspondence between periods of audio silence. The token 'ε' is used for repetitions, and the underscore "_" is used for spaces between words (see figure 4).

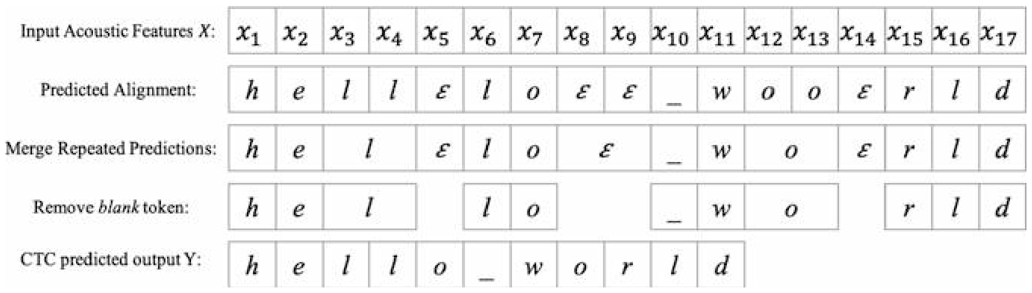

Input Acoustic Features X: x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10} x_{11} x_{12} x_{13} x_{14} x_{15} x_{16} x_{17}
Predicted Alignment: h e l l ε l o ε ε _ w o o ε r l d
Merge Repeated Predictions: h e l ε l o ε _ w o ε r l d
Remove *blank* token: h e l l o _ w o r l d
CTC predicted output Y: h e l l o _ w o r l d

Figure 4: CTC alignment for an input X and output $Y = [h, e, l, l, o, \_, w, o, r, l, d]$

This approach yields a 1:1 alignment between the elements of X and Y. On top of that, introducing the 'ε' character generates the chance of obtaining many predicted alignments that lead to the same output. For example:

[f, ε, r, ε, e, ε, e] –> free
[f, r, ε, e, ε, e, ε] –> free

The CTC algorithm is "alignment-free" meaning it does not require them to function as these are only used to calculate the probabilities of possible alignments.

Then it produces an output distribution of all possible Ys. These can be used to infer the probability of some concrete output, Y. The conditioned probability, $P(Y|X)$, is calculated with the sum of all the possible alignments between X and Y (see figure 5).

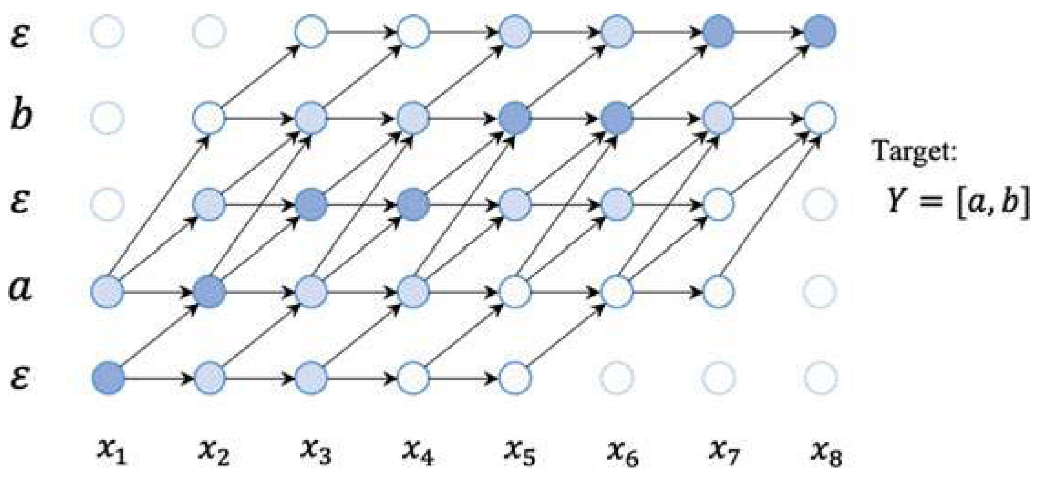


Figure 5: A graph showing the valid set of paths from the target sequence to an output. We have two possible initial states: ε or a, as well as two possible final states: ε or b [15]

This graph creates connections between the different tokens possible to be obtained in the result and denotes the different paths that could be created towards obtaining output from the input sequence. The ε tagged nodes should only be able to traverse to themselves or the next character of the sequence, 'a' in the case of the graph at x_1, figure 5. The same happens with the character 'a' at x_2, only in this case do we have 3 alternatives: travel to itself, to ε or to the next character in the sequence, 'b' in this case.

Taking the equations and the explanation from [15]:

Mathematically, we can define the conditional probability of single alignment α_t, as the product of each state in the sequence:

$$P(\alpha|X) = \prod_{t=1}^{T} P(\alpha_t|X)$$

All paths are considered mutually exclusive, so we sum the probability of all alignments, giving the conditional probability for a single utterance (X, Y):

$$P(Y|X) = \sum_{A \in A_{X,Y}} \prod_{t=1}^{T} P(\alpha_t|X)$$

where $A_{X,Y}$ is the set of valid alignments. Dynamic programming is used to improve the computation of the CTC loss function. By supplying blank tokens around each label in the sequence, the paths can be easily comparable and merged when they reach the same output at the same time step.

Combining everything gives the loss function for CTC:

$$L_{CTC}(X,Y) = -\log \sum_{a \in A_{X,Y}} \prod_{t=1}^{T} P(a_t|X)$$

The gradient for backpropagation can be computed for each time step from the probabilities at each frame.

CTC assumes conditional independence between each time step in that the output at each time step is independent of the previous time steps. Although this property allows for frame-wise gradient propagation, it limits the ability to learn sequential dependencies. Using a language model alleviates some of the issues by providing a word or n-gram context.

1.5 Mozilla DeepSpeech

In 2014, a group of researchers from Baidu Research published a paper [12] proposing the application of revolutionary new AI techniques to the field of Speech Recognition. This work set the basis for the Mozilla DeepSpeech project, although the Mozilla DeepSpeech engine currently differs in many ways from that initial document.

The described approach explains that traditional Speech Recognition Systems are composed of intricate pipelines of components that have to be meticulously engineered. These parts would include, among others, the following: models for background noise, speaker voice identification, or reverberation.

Instead of using these sophisticated data processing stages, this paper describes an end-to-end[1] speech system called "DeepSpeech". This device achieves higher performance than those preceding it while also being quite simple.

The results obtained are possible thanks to the usage of great amounts of hours of data and a lot of processing power (specifically GPUs) for the training phase. A large Recurrent Neural Network (RNN) (See section 1.3) is used as the basis for the trained model.

[1]End-to-end means that data is not processed by a large engineered pipeline. Instead, it is the model that analyzes the data to produce a transcription output. (Only with a few exceptions to optimizations and output correction)

The key here is that the model leverages deep learning algorithms. This enables the user to process huge amounts of data while maintaining good performance and avoiding the usage of complex data processing pipelines.

By choosing this type of Neural Network, the authors of the paper can map the training process really well to GPUs. They also used a novel model partition scheme that helped improve parallel execution. The paper presents a model trained with both collected and synthesized data. This paper also proposes a process for assembling large quantities of labelled speech data. It also protects the outcomes from the distortions present in the data and that shall be handled by the model itself. The result from this recipe is a well-trained system that transcribes audio to text and that is robust to realistic noise and speaker variation, including the Lombard Effect[2].

1.5.1 RNN Training Setup

As we explained earlier, the core of the DeepSpeech system is a Recurrent Neural Network (RNN). In our case, it is not trained to ingest English audio files, but Spanish instead. So it produces Spanish transcriptions.

Proceeding with the explanation included in the paper [12], we will describe the model as follows: Given a training set $X = \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), ...\}$, we will sample 2-tuples containing two values: x being each utterance, and y being the label of such utterance. Each of these x values, $x^{(i)}$, is a time-series of length $T^{(i)}$ where every time-slice is a vector of audio features, $x_t^{(i)}$, with $t = 1, ..., T^{(i)}$. The system uses spectrograms as their audio features, so $x_{t,p}^{(i)}$ denotes the power of the p'th frequency bin in the audio frame at time t. The objective of the proposed system is to output a sequence of character probabilities for each single transcription y given an input x. This sequence is described as $\hat{y}_t = \mathbb{P}(c_t|x)$ where $c_t = \{$a, b, c, ..., z, ñ, á, é, í, ó, ú, $blank, space\}$. This alphabet has been expanded from the one presented in [12] as we need to include the 'ñ' character as well as the acute-accented vowels. Also, we do not need apostrophes.

The RNN model proposed by the paper [12] is composed of 5 layers of hidden neurons (see figure 6). Given an input utterance x, the hidden neurons located at layer l are denoted $h^{(l)}$. As a convention, $h^{(0)}$ is the input layer. The first three layers of the model are not recurrent. For the first layer, at each time t, the output depends on the spectrogram frame x_t along with a context of C frames on each side (these frames represent the input audio fragments that affect such output). The paper [12] proposes that $C \in \{5, 7, 9\}$ would be their setting for the experiments.

The remaining non-recurrent layers operate on independent data for each time step. Thus, for each time t, the first 3 layers are computed by:

$$h_t^{(l)} = g(W^{(l)} h_t^{(l-1)} + b^{(l)})$$

where $g(z) = min\{max\{0, z\}, 20\}$ is the clipped rectified-linear (ReLu[3]) activation function and $W^{(l)}, b^{(l)}$ are the weight matrix and bias parameters for layer l, respectively.

Recurrency is only introduced in the fourth layer as a bi-directional recurrent layer. This layer includes two sets of hidden units: a set with forward recurrence $h^{(f)}$, and a set

[2]The Lombard Effect [18] is encountered in noisy environments when a speaker actively (but involuntarily) modifies the tone and/or pitch of their voices to overcome such background noise.

[3]The ReLu units are clipped to keep the activations in the recurrent layer from exploding; in practice, the units rarely saturate at the upper bound.

with backward recurrence $h^{(b)}$. Their respective outputs are calculated as follows:

$$h_t^{(f)} = g(W^{(4)}h_t^{(3)} + W_r^{(f)}h_{t-1}^{(f)} + b^{(4)})$$

$$h_t^{(b)} = g(W^{(4)}h_t^{(3)} + W_r^{(b)}h_{t+1}^{(b)} + b^{(4)})$$

Due to the recurrent structure $h^{(f)}$ must be computed sequentially from $t = 1$ to $t = T^{(i)}$ for the i'th utterance, while the units $h^{(b)}$ must be computed sequentially in reverse order from $t = T^{(i)}$ to $t = 1$.

The fifth layer takes both the forward and backward units as inputs $h_t^{(5)} = g(W^{(5)}h_t^{(4)} + b^{(5)})$ where $h_t^{(4)} = h_t^{(f)} + h_t^{(b)}$. The output layer is a standard softmax function that yields the predicted character probabilities for each time slice t and character k in the alphabet:

$$h_{t,k}^{(6)} = \hat{y}_{t,k} \equiv \mathbb{P}(c_t = k|x) = \frac{exp(W_k^{(6)}h_t^{(5)} + b_k^{(6)})}{\sum_j exp(W_j^{(6)}h_t^{(5)} + b_j^{(6)})}$$

As soon as we have computed the character predictions, we can calculate the CTC loss (see section 1.4.1) so we can measure the prediction error. As the training process takes place, we can evaluate the gradient for the network outputs given the ground-truth character sequence y (a.k.a. the label of the utterance). From this point, computing the gradient for all of the model parameters may be done via back-propagation through the rest of the network.

The model in the paper uses Nesterov's Accelerated gradient method for training. But for our specific training, we use Adam's method. This is not my decision, it is pure convenience. The default project training setup uses ADAM's for optimization. The client allows for the configuration of the optimizer, but this was left unchanged.

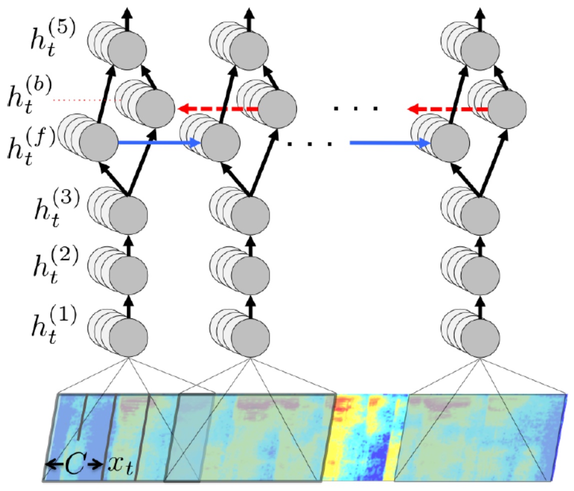


Figure 6: Structure of the RNN model proposed in [12]

1.5.1.1 Nesterov's Accelerated Gradient Method

This method is a variant of a Stochastic Gradient Descent (SGD) optimizer based on momentum that tries to "look ahead" in the calculations of the gradient using the accumulated gradient up to the current location. A standard momentum takes a big jump in the direction of the updated accumulated gradient. Whereas Nesterov's momentum does the same thing but uses the previously accumulated gradient. In addition, it corrects the gradient, based on where it ended up.

1.5.1.2 Adam's Optimization Method

Deriving its name from the term Adaptive Moment Estimation, this optimization method combines the advantages of two extensions of the SGD method: Adaptive Gradient Algorithm (AdaGrad) and Root Mean Square Propagation (RMSProp). This algorithm is commonly adopted for deep learning in applications of Computer Vision or Natural Language Processing. It is also known for being fast at obtaining results as well as memory effective.

1.5.2 Regularization

There is a problem with training almost any Machine Learning model. It is called 'Overfitting', and it means the model has almost perfectly learned the small details of the training data, losing its ability to generalize. It is really good at fitting the training data but is unable to perform correctly with new, unseen data. This phenomenon is a constant tendency of any good model trained to fit the training dataset.

To try and prevent this, the researchers apply what is known as dropout. This prevents some of the training samples to affect the network's weights and biases. The dropout rate used in the paper is between 5% and 10% and it is applied to the feed-forward layers but not to the recurrent hidden activations [12].

1.5.3 Language Model

Starting from just the RNN model, the system can already produce readable character-level transcriptions given enough training data to start with. These predictions obtained by the raw model are often just perfect and match the input label. However, some errors that make the output labels inaccurate by some pronunciation errors (in English vowels tend to be the problem). Well, these issues can be solved using a language model.

The DeepSpeech project uses an N-Gram model to enhance the predictions of the network. These language models are relatively easy to train from unlabelled text corpora.

The project used a total of 220 million English phrases, supporting a total of 495,000 words. The language model used was trained using the KenLM toolkit [12].

Given the output of $\mathbb{P}(c|x)$ of the RNN a search is performed aimed at finding a sequence of characters $c_1, c_2, \ldots$ that is most probable according to the RNN's output and the language model used. More precisely, the aim is to find a sequence c that maximizes the combined goal:

$$Q(c) = \log(\mathbb{P}(c|x)) + \alpha \log(\mathbb{P}_{lm}(c)) + \beta word\_count(c)$$

where α and β are tunable parameters (set by cross-validation) that control the trade-off between the RNN, the language model constraint and the length of the sentence. The term $\mathbb{P}_{lm}$ denotes the probability of the sequence c according to the N-gram model. This objective is maximized using a highly optimized beam search algorithm, with a typical beam size in the range 1000-8000. Similar to the approach described by Hannun et al. [12].

1.5.4 Optimizations

A quality-of-life feature that this model has is that it can train new models in a fast way. This is achieved through many implementations and design decisions that make the network amenable to high-speed execution.

The network model is fully connected in almost every layer, thus efficient execution is critical to make it worth using. The training is also made faster by using multiple GPUs.

On top of that, some more specifics are explained below on how this system was optimized in the paper.

1.5.4.1 Data parallelism To process data efficiently, DeepSpeech makes the GPU process many examples in parallel. This is done in the usual way by concatenating many examples into a single matrix. Instead of performing single matrix-vector multiplication in the recurrent layer, many are performed in parallel computing creating a bigger matrix that holds several training examples. This is done because the GPUs are most efficient when the matrices are wide enough. Up to the limits of the GPU's memory.

1.5.4.2 Model parallelism Data parallelism yields training speedups for modest multiples of the minibatch size (e.g., 2 to 4), but faces diminishing returns as batching more examples into a single gradient update fails to improve the training convergence rate. Simply put, if we processed double the examples on double de GPUs, we would not double the speedup in training.

1.5.4.3 Striding DeepSpeech has worked to minimize the running time of the recurrent layers of the RNN model since these are the hardest to parallelize. As a final optimization, the recurrent layers are shortened by taking "strides" of size 2 in the original input so that the unrolled RNN has half as many steps.

2 Materials and Methods

To obtain a model capable of transforming text into audio using a neural network, we first need data. Examples that such networks will use to learn how to transcribe audio into text.

There is a need for this data to have decent quality, otherwise, the output model will not be as useful as expected. This means that for us to generate high-quality transcriptions, we need to feed in high-quality audio and transcriptions into the system. This is what I was referring to as **supervised learning**.

To extract knowledge from the data, we follow a process composed of a series of stages. These can be named as:

1. **Selection** of necessary data for our experiment or project.
2. **Cleaning**. Creates a subset of the input data (optionally) transforming it in some way like removing unwanted parts, reducing noisy parts or replacing missing data for instance.
3. **Coding**. Meaning to create an organized data structure for the model to ingest.
4. **Learning** from the dataset to create a model that represents whatever is in our universe of discourse.
5. **Inference** to generate new information based on the model and unknown data.

These activities work towards the final objective of creating new knowledge based on the processed data.

2.1 Selection of datasets

For this project, I had been provided with a series of datasets including raw audios and their transcriptions for me to work with. Some datasets were obtained from sources either no longer possible to find or that are unknown to this day.

2.1.1 Summary

The charts shown in figures 7 and 8 show the number of audios in each dataset used in the project with regards to total audio hours and the total number of elements.

In both of them, we can appreciate that the slice for CommonVoice is predominant, being the heaviest and biggest dataset of all of them.

2.1.1.1 Hours of audio per dataset

Note: For the sake of a clean-looking chart, the amounts of hours for each dataset have been rounded to the closest unit.

2.1.1.2 Number of audio clips per dataset

Note: Due to the limitations of the page size, the number of elements is represented with a percentage and the actual number is written beside the legend.

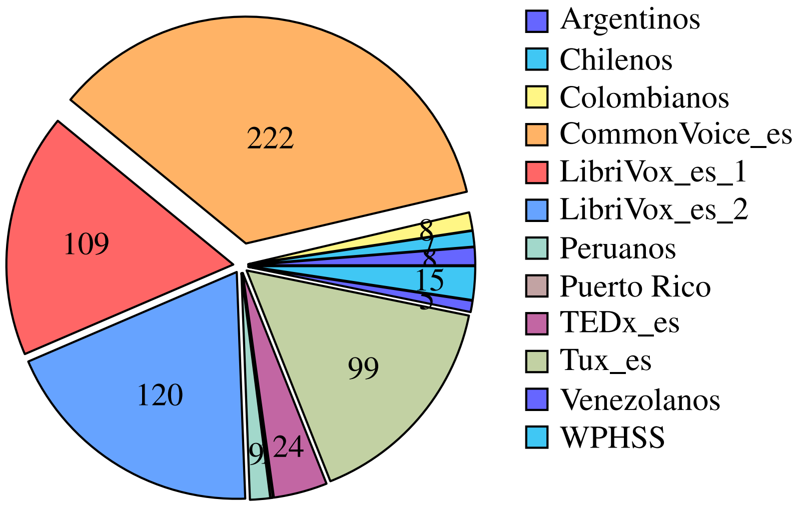


Figure 7: A pie chart showing the distribution of audio hours per dataset in the training data

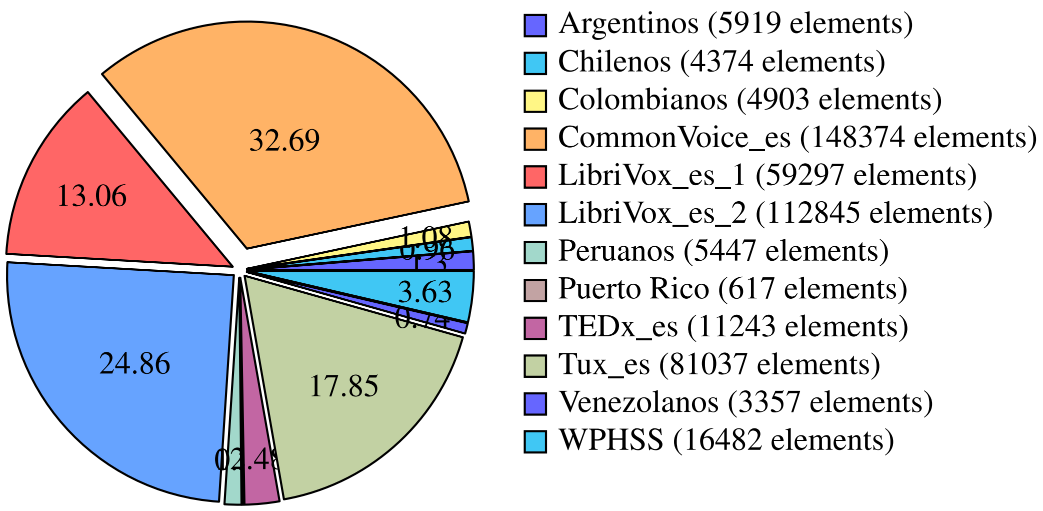


Figure 8: A pie chart showing the distribution of audio elements per dataset in the training data

2.1.2 Details

Let us list all the datasets used in the process, some information about them and some attribution to the creator(s) if found:

- Argentinos
  - **Overall description of the dataset:** This dataset contains mostly female speech and includes a small portion of around 200 samples of small weather reports.
  - **Audio quality:** The overall quality is not optimal. Plus, there is a bit of extra silence at the end of the clips.
  - **Completion/sparsity of the data:** This dataset contains a total of 5919 elements. It is divided into three groups with male speakers, female speakers and weather messages. This last one is subdivided into Castellian Spanish accent

and Argentinian Spanish accent. Female speech is predominant in this dataset with a total of 3800+ elements.
  - **total approximate duration:** 8.20 hours of speech.
  - **Reference and attribution:** None found.

- Chilenos
  - **Overall description of the dataset:** This dataset, along with some others of this list represents a part of the Latino-American variety of accents of Spanish. Not a very big dataset, gives us a small insight into the Chilean accent.
  - **Audio quality:** The audio quality is not optimal. Plus the samples contain way too much silence at the end of the clips which could lead to malfunction of the model.
  - **Completion/sparsity of the data:** This dataset contains a total of 4374 elements. The dataset is divided into two groups for male and female speakers. These groups contain approximately half of the audio samples for each gender.
  - **total approximate duration:** 7.14 hours of speech.
  - **Reference and attribution:** None found.
- Colombianos
  - **Overall description of the dataset:** This dataset represents the Colombian Spanish accent with clear short-sentences speech.
  - **Audio quality:** The quality is not bad but it is not optimal. However, the audio is clear and it seems that the samples do not contain that much silence.
  - **Completion/sparsity of the data:** This dataset contains a total of 4903 elements. The dataset is divided into two groups for male and female speakers. These groups contain approximately half of the audio samples for each gender.
  - **total approximate duration:** 7.58 hours of speech.
  - **Reference and attribution:** None found.
- CommonVoice_es
  - **Overall description of the dataset:** This dataset provides the world with a huge open and multi-language database of voices. Anyone can use this database to train AI models that can be used in applications that use the human voice as an interface. This database is expanded each day and to date, it has around 700 hours of Spanish speech (2000+ hours for English speech, for instance) from which 380 hours have been validated by human beings. This allows you to work with really good-quality speech transcription that helps build good-performing AI models for SST or TTS systems. The possibilities are limitless.
  - **Audio quality:** The quality in this dataset is far better than the other ones. The recordings are much clearer and do not contain static.
  - **Completion/sparsity of the data:** This dataset contains a total of 148374 elements. Some faulty audios came corrupted from the download site and therefore cannot be used.
  - **total approximate duration:** 221.82 hours of speech.
  - **Reference and attribution:** Common Voice Mozilla
- LibriVox_es_1
  - **Overall description of the dataset:** This dataset is really difficult to read because the data is nested in a lot of folders. Difficult to organize but is quite

big.
- **Audio quality:** The audio files seem loud and clear, slowly read so every word is distinguished. The audio quality could be better.
- **Completion/sparsity of the data:** The dataset includes many folder separations for various speakers and includes a pair of CSV files to separate clean audio from the other clips. This dataset contains a total of 59297 elements.
- **total approximate duration:** 108.58 hours of speech.
- **Reference and attribution:** LibriVox, Acoustical liberation of books in the public domain.

- LibriVox_es_2
  - **Overall description of the dataset:** This dataset features a total of 17 speakers without counting the collaborative audiobooks. This audio collection has been cut by the windows speech recognition using the source text as grammar, then validated with the DeepSpeech Spanish model.
  - **Audio quality:** The audio files seem loud and clear, slowly read so every word is distinguished. The audio quality could be better.
  - **Completion/sparsity of the data:** The dataset includes two CSV files to separate clean audio from the other clips. This dataset contains a total of 112845 elements.
  - **total approximate duration:** 119.64 hours of speech.
  - **Reference and attribution:** LibriVox, Acoustical liberation of books in the public domain.

- Peruanos
  - **Overall description of the dataset:** This dataset contains Peruvian accent speech. The speaker speaks slowly so each word gets understood clearly. It also has the right amount of silence on both ends of the audio.
  - **Audio quality:** Although the audio is loud and clear. The quality is not that good.
  - **Completion/sparsity of the data:** The dataset is divided into two groups of men and women speech. With a total of 5447 elements with close to half the elements for each group.
  - **total approximate duration:** 9.22 hours of speech.
  - **Reference and attribution:** None found.

- Puerto_rico
  - **Overall description of the dataset:** This small dataset contains short sentences with a Puerto Rico accent.
  - **Audio quality:** The quality is not that of a studio, but good enough to train for normal phone-like speech. However, the audio clips contain excess silence at the end that could mess with the AI model.
  - **Completion/sparsity of the data:** The dataset is really small compared to the other ones, containing just 617 elements.
  - **total approximate duration:** 1 hour of speech.
  - **Reference and attribution:** None found.

- TEDx_es
  - **Overall description of the dataset:** The dataset contains speech fragments of lecturers speaking during TEDx locally organized events.

  - **Audio quality:** The audio is a little bit crunchy and its quality is not so good.
  - **Completion/sparsity of the data:** The dataset includes a lot of speech files, a total of 11243 elements.
  - **total approximate duration:** 24.49 hours of speech.
  - **Reference and attribution:** TEDx is a program of local, self-organized events that bring people together to share a TED-like experience.

- Tux_es
  - **Overall description of the dataset:** This dataset comes from a public domain audiobook page. It is divided into two groups: "Other" which is not validated, and "Valid" which has been validated by Mozilla's DeepSpeech model. It contains audios reading sentences written in books.
  - **Audio quality:** The audios are loud and clear. Some of them are too tightly cut so that audio silence is not present on either end of the file.
  - **Completion/sparsity of the data:** The "usable" data is halved since around 50h of speech are not validated. The other half is also cleaned in a CSV file that removes around 1000 files.
  - **total approximate duration:** 99.43 hours of speech.
  - **Reference and attribution:** LibriVox, Acoustical liberation of books in the public domain

- Venezolanos
  - **Overall description of the dataset:** This dataset includes audio from male and female people with (supposedly) Venezuelan accents.
  - **Audio quality:** The audio has not been recorded inside a studio so the quality is not that good. They were probably recorded with a cheap microphone.
  - **Completion/sparsity of the data:** The dataset is not that big. It includes a total of 3357 audio files, half the files are from women and half the files are from men.
  - **total approximate duration:** 4.81 hours of speech.
  - **Reference and attribution:** None found.

- West Point Heroico Spanish Speech (WPHSS)
  - **Overall description of the dataset:** This next piece of text is taken from the website where the data is available. "This file contains documentation on the West Point Heroico Spanish Speech, Linguistic Data Consortium (LDC) catalogue number LDC2006S37 and isbn 1-58563-391-7. This corpus was designed and collected by staff and faculty of DFL and CTELL to develop acoustic models for speech recognition systems. The U.S. government uses these systems to provide speech-recognition enhanced language learning courseware to government linguists and students enrolled in various government language programs. Additionally, parts of this corpus were designed to model question/answer dialogues for use in domain-specific speech-to-speech translation systems".
  - **Audio quality:** There were some faulty audios and some recordings contained static. Plus, some questions are not answered by some speakers.
  - **Completion/sparsity of the data:** Complete enough to create a solid dataset, but complex structure when merging it into a unique dataset.
  - **total approximate duration:** 14.82 hours of short speech fragments.

- **Reference and attribution:** Linguistic Data Consortium (LDC).

2.2 Preprocessing

Cleaning and coding the datasets into a unique piece of data is a tough procedure. Model quality depends on the way these tasks are performed.

2.2.1 Segmentation of the audio files

Based on the content laid out in section 2.1, we ended up with a large number of datasets to process for training. Not all of these made it to the final training dataset.

As could be expected, each dataset had a specific folder structure so they could not be processed in the same way with the same code base. To overcome this, I created a Python script that included a class to represent each of the datasets. They were all inheriting from a base class that implemented a set of functions to manage the data uniformly to be later processed by another script explained in the next section (2.2.2).

This script is highly coupled to the folder structure to allow the next script to process them using a common interface.

Plus, almost all the datasets contain data index files. These files are often tabular-separated-values (.tsv) or comma-separated-values (.csv). Explaining the columns in the data seem of little purpose to me since these can be read in the data files and the Python script fragments listed below. The meaning is easy to deduce if both are consulted. Just as a simple summary, all of them contain a column to indicate the name of the file and often the path to the file inside the dataset folder and a column with the text label. There are also other columns in some datasets, but these are either useless or duplicated.

The file paths were often not composed to the dataset directory and had to be created manually through the script and scanning the folders to deduce the structure that the paths needed to have. Doing so was necessary in the case the user decided to merge the datasets. These would be moved to a new folder and so, the original paths to the files would break due to the different datasets present and the new folder structure created around them.

2.2.1.1 Adapter Design Pattern Figure 9 is an image of the Adapter Design Pattern represented using UML taken from [11], that sums up the basic structure of the pattern.

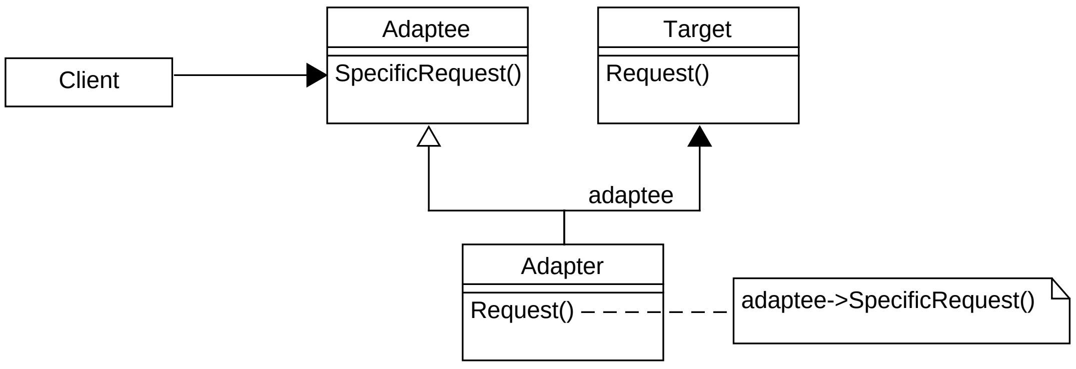


Figure 9: A UML class diagram showing the relationships between the participants in the Adapter or Wrapper Design Pattern

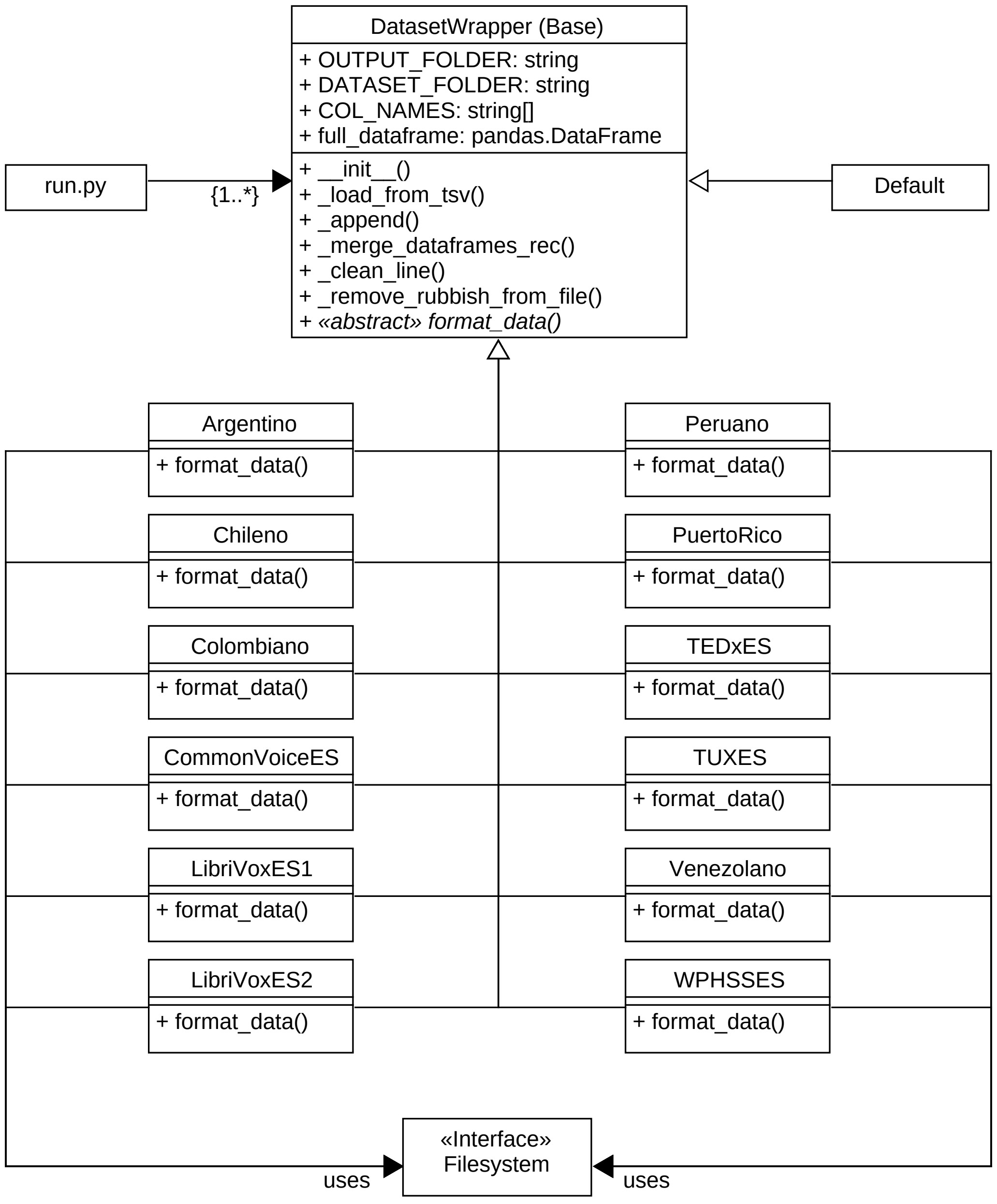


Figure 10: A UML class diagram showing the participants in the implementation of the Adapter Design Pattern applied to the project

All the subclasses implementing the DatasetWrapper class shown in figure 10 are also implementing the format_data() function. In essence, with the arrows pointing to the Filesystem Interface, I mean that each of the classes interfaces the filesystem in a specific way the dataset needs. Meaning these load the data from the index files (*.tsv) or manipulate it as needed, so the rest of the functions work decoupled from this mess of files and folder juggling.

It also means that, if a new dataset is added in the future, nothing needs to be changed. We just need a new subclass, make it implement the format_data() and call its constructor from the main script.

2.2.1.2 Adapter classes in the script Now, we will explain the Python classes that implement each interface to the filesystem to read the training files for each dataset. This process will include segments of the whole 'datasets.py' script under 'final-degree-project/code_repos/recogida-audios/dataset-processing'.

2.2.1.2.1 General knowledge about the script To work efficiently with all the tuples in the training files, we used the Data Science library 'pandas'. This library allows us to use an object named DataFrame which has powerful functions to manage row-like data.

Moreover, I am aware that coupling the filesystem paths and the script was not a good idea. Knowing this, an improved and uncoupled solution could be crafted using os.walk() and adapting the folder structure by hand. I opted out of those two solutions. Either two would require almost the same time as following the coupled approach. Anyways, if the project folder structure remains the same when executing the script, there should be no problem.

2.2.1.2.2 Base Adapter class This part of the script includes the definition of the base class that implements general-use functions to manipulate the datasets. Here is a breakdown of the functions implemented by the class:

- **__init__():** The object constructor. Receives a parameter 'folder_name' that contains the name of the folder that contains the dataset this class instance represents.
- **_load_from_tsv():** This function is in charge of reading the contents of the dataset index files into a pandas DataFrame object. These files are often ending in .tsv (meaning values are separated by a tabular char '\t'). These could also be .csv files (the value-separating character is a comma ','). Let us briefly explain the parameters of the function:
  - **tsv_file_path:** The path leading to the data index file.
  - **path_prefix:** The path prefix to access the folder.
  - **override_names=True:** If this is set to true, the function will generate a DataFrame with a custom list of headers rather than the default one. This list of headers is set when instantiating the class (will be overridden by child classes).
  - **separator='\t':** The character separating data in the files in case it is changed. Defaults for tabular character for .tsv files.
  - **skip_rows=0:** A number of rows to skip reading from the file, in case it is needed.
  - **path_suffix=".wav":** The extension suffix of the audio files, in case it needs to be changed.
- **_append():** Receives two datasets as input and appends the first to the second, ignoring indices to not break data structure and continuing the numbered series.
- **_merge_dataframes_rec():** This function merges recursively all the DataFrame objects found in the 'dataframes' parameter into a single one.
  - **dataframes:** Expects a list of DataFrame objects to merge all of them recursively into a single one.
  - **current_index:** A parameter to traverse the list and the variable used for the recursion stopping condition.

- **_clean_line():** This function removes some characters from the input string and returns the clean string.
- **_remove_rubbish_from_file():** This function cleans the lines from the input file using the clean_line function.
- **@abstractmethod format_data():** This abstract method is to be implemented by the child classes because it is the part that adapts the interface of the class to the specific dataset we are working with. Here we should import each data index file and manipulate it so all are merged into one.

```python
class Base(ABC):
    def __init__(self, folder_name):
        super().__init__()
        self.OUTPUT_FOLDER = path.join("/", "home", "danifinca", "stt", "data", folder_name)
        self.DATASET_FOLDER = path.join(STT_DATASETS_FOLDER, folder_name)
        self.COL_NAMES = ["path", "sentence"]
        self.full_dataframe = None

    def _load_from_tsv(self, tsv_file_path, path_prefix, override_names=True,
        separator="\t", skip_rows=0, path_suffix=".wav"):
        tsv_file_path = self._remove_rubbish_from_file(tsv_file_path, separator)
        if override_names:
            dataframe = pd.read_csv(
                tsv_file_path,
                sep=separator,
                skiprows=skip_rows,
                names=self.COL_NAMES,
                dtype={col_name: str for col_name in self.COL_NAMES}
            )
        else:
            dataframe = pd.read_csv(
                tsv_file_path,
                sep=separator,
                dtype={col_name: str for col_name in self.COL_NAMES}
            )
        dataframe["path"] = path.join(
            self.DATASET_FOLDER, "clips", path_prefix
        ) + dataframe["path"] + path_suffix
        dataframe["client_id"] = [uuid4() for i in range(dataframe.shape[0])]
        dataframe = dataframe[["client_id", "path", "sentence"]]
        return dataframe

    def _append(self, dataframe1, dataframe2):
        return dataframe1.append(
            dataframe2,
            ignore_index=True,
            verify_integrity=True,
            sort=True
        )

    def _merge_dataframes_rec(self, dataframes, current_index):
        if len(dataframes) < 1:
            raise ValueError("No dataframes were provided to merge"
   )
        if len(dataframes) == 1:
            return dataframes[0]

        last_two_dfs = current_index == len(dataframes) - 2
        return (self._append(
            dataframes[current_index],
            dataframes[current_index + 1]
                if last_two_dfs
                else self._merge_dataframes_rec(dataframes,
   current_index + 1)
            )
        )

    def _clean_line(self, line, separator):
        return re.sub("¿¡ºª°’[\"\’?!]", "", line)

    def _remove_rubbish_from_file(self, tsv_file_path, separator):
        clean_filename = path.splitext(tsv_file_path)
        clean_filename = clean_filename[0] + "_clean" +
   clean_filename[1]
        lines = []
        with open(tsv_file_path, "r") as input_file:
            lines = input_file.readlines()
            lines = [
                self._clean_line(line, separator)
                for line
                in lines
            ]
        with open(clean_filename, "w") as output_file:
            output_file.writelines(lines)
        return clean_filename

    @abstractmethod
    def format_data(self):
        raise NotImplementedError((
            "You cannot call this method because this is a base
   class"
            " for constructing dataset readers."
        ))
```

Listing 1: A fragment of the datasets.py script featuring the base class for the adapter

2.2.1.2.3 Default Adapter class This class was created to allow the main preprocessing script to merge all datasets into one. This one receives an extra parameter in its constructor for the client to give it a specific name. It does not represent any input dataset. It translates into the output dataset if the user instructs the main script to merge the datasets.

```
¿¡ºª°’
class Default(Base):
    def __init__(self, folder_name):
        super().__init__(folder_name)
        self.DATASET_FOLDER = path.join(STT_DATASETS_FOLDER,
   folder_name)
```

```
        self.COL_NAMES = ["path", "sentence"]

    def format_data(self):
        self.full_dataframe = pd.DataFrame()
        return self.full_dataframe
```
Listing 2: A fragment of the datasets.py script featuring the Default class for the adapter

2.2.1.2.4 Argentino Adapter class One of the multiple datasets that include specific accent audio files. Initializing the folder name to ’argentinos’, this dataset is composed of 4 separate index files.

- weather_es_ar, audio files in a weather-related context with Argentinian accent,
- weather_es_es, same as weather_es_ar but this time is Castillian accent,
- line_index_female, audio files of female speakers,
- line_index_male, audio files of male speakers.

```
¿¡ºª◦’
class Argentino(Base):
    def __init__(self):
        super().__init__("argentinos")

    def format_data(self):
        weather_es_ar = self._load_from_tsv(
            path.join(self.DATASET_FOLDER, "
    es_ar_line_index_weather.tsv"),
            path.join("es_weather_messages", "es-ar") + path.sep
        )
        weather_es_es = self._load_from_tsv(
            path.join(self.DATASET_FOLDER, "
    es_es_line_index_weather.tsv"),
            path.join("es_weather_messages", "es-es") + path.sep
        )
        line_index_female = self._load_from_tsv(
            path.join(self.DATASET_FOLDER, "line_index_female.tsv")
    ,
            "es_ar_female" + path.sep
        )
        line_index_male = self._load_from_tsv(
            path.join(self.DATASET_FOLDER, "line_index_male.tsv"),
            "es_ar_male" + path.sep
        )
        self.full_dataframe = self._merge_dataframes_rec([
            weather_es_ar, weather_es_es, line_index_female,
    line_index_male
        ], 0)
        return self.full_dataframe
```
Listing 3: A fragment of the datasets.py script featuring the Argentino class for the adapter

```
-> line_index_female.tsv
[...]
arf_05679_01463815283 Para la ícada del cabello, tengo un nuevo
   úchamp
arf_05223_00524892324 Los áhmsters comen zanahorias
arf_07973_01243309438 ¿Me épods mandar fotos de la pileta?
[...]
```

```

-> line_index_male.tsv
[...]
arm_09697_00831674848 Para la ícada del cabello, tengo un nuevo
    ójabn
arm_01523_01987826609 ¿éQu color favorito es el áms popular?
arm_08784_00917516295 Las ámquinas de escribir antiguas pueden ser
    muy caras.
[...]

-> es_ar_line_index_weather.tsv
[...]
arf_02485_00047151674 Hace doce grados con sol
arf_02485_00146903919 Hace trece grados con sol
arf_02485_00204623004 Hace doce grados y áest nublado
[...]

-> es_es_line_index_weather.tsv
[...]
esw_03397_00872842342 Hay diecisiete grados con sol
esw_02484_00327933988 Hay quince grados y llueve
esw_03397_01885457045 Hay diecinueve grados y áest nublado
[...]
```

Listing 4: Small sample of the rows contained in the input data index files of the Argentino dataset

2.2.1.2.5 Chileno Adapter class This other Spanish accent variant contained just two files. One referenced audio files from women and the other referenced audio files from men.

```
¿¡ºª°,
class Chileno(Base):
    def __init__(self):
        super().__init__("chilenos")

    def format_data(self):
        line_index_female = self._load_from_tsv(
            path.join(self.DATASET_FOLDER, "line_index_female.tsv")
    ,
            "es_cl_female" + path.sep
        )
        line_index_male = self._load_from_tsv(
            path.join(self.DATASET_FOLDER, "line_index_male.tsv"),
            "es_cl_male" + path.sep
        )
        self.full_dataframe = self._merge_dataframes_rec([
            line_index_female, line_index_male
        ], 0)
        return self.full_dataframe
```

Listing 5: A fragment of the datasets.py script featuring the Chileno class for the adapter

```
-> line_index_female.tsv
[...]
clf_09334_01278378087 La vigencia de tu tarjeta es de ocho meses
clf_09697_00015596584 Tranquilo va a estar todo bien
```

```
clf_09697_01505655474 Me gusta mucho caminar por el campo y tomarle
    fotos a la naturaleza
[...]

-> line_index_male.tsv
[...]
clm_08421_01719502739 Es un viaje de negocios solamente voy por una
    noche
clm_02436_02011517900 Se usa para incitar a alguien a sacar el
   mayor provecho del dia presente
clm_09697_00628052255 Los ñnios tienen mucha óimaginacin
[...]
```

Listing 6: Small sample of the rows contained in the input data index files of the Chileno dataset

2.2.1.2.6 Colombiano Adapter class This dataset contained two files, men and women audio files, respectively.

```
¿¡ºªº’
class Colombiano(Base):
    def __init__(self):
        super().__init__("colombianos")

    def format_data(self):
        line_index_female = self._load_from_tsv(
            path.join(self.DATASET_FOLDER, "line_index_female.tsv")
,
            "es_co_female" + path.sep
        )
        line_index_male = self._load_from_tsv(
            path.join(self.DATASET_FOLDER, "line_index_male.tsv"),
            "es_co_male" + path.sep
        )
        self.full_dataframe = self._merge_dataframes_rec([
            line_index_female, line_index_male
        ], 0)
        return self.full_dataframe
```

Listing 7: A fragment of the datasets.py script featuring the Colombiano class for the adapter

```
-> line_index_female.tsv
[...]
cof_02436_01372133479 Quiero saber équ áest pasando en Veracruz.
cof_03397_01983407356 ¿Puedes revisar si hay alguna tienda departo
   cerca de la casa de mis ápaps?
cof_07508_01601808212 ¿Quieres que revise tu ónmina y los ódepsitos
   que han hecho en el último mes?
[...]

-> line_index_male.tsv
[...]
com_03349_00001764679 El íjeroglfico tiene un pez amarillo
com_07508_02061155104 ¿Hay úalgn trabajo en particular que le haya
   dado como el salto a la fama por ías decirlo?
com_02121_01523018450 Los zapatos se estropearon con la lluvia
   torrencial de ayer
```

```
[...]
```

Listing 8: Small sample of the rows contained in the input data index files of the Colombiano dataset

2.2.1.2.7 CommonVoiceES Adapter class This dataset was the one with the best quality and number of files. This one is an ever-growing dataset, so as of today, this version we are using is probably only a subset of the current release. Moreover, this dataset's files contain more columns than needed for this project. There are columns such as down_votes or up_votes that are not needed. Neither is the client_id column, but this one needs to be present in the output format because the import script to run before training expects this column in the input data. There are also columns with missing data, but as these are not needed, we are not affected by this issue in the data.

This dataset contains several index files:

- dev: Audio files to use for validation,
- invalidated: Audio files invalidated by the community. Meaning the label associated with the audio does not correspond with what the speaker says,
- other,
- test: Audio files to use for testing,
- train: Audio files to use for training,
- validated: Audio files validated by the community. That is, the audio label and the speaker's speech correspond.

This file separation is convenient because this dataset is already prepared to be used as-is for training DeepSpeech. In fact, this dataset is used in the engine's documentation when creating an example of commands to execute.

```python
¿¡ºªº,
class CommonVoiceES(Base):
    def __init__(self):
        super().__init__("cv_es")

    def _factor_load(self, filename):
        return self._load_from_tsv(
            path.join(self.DATASET_FOLDER, filename),
            "",
            override_names=False,
            path_suffix=""
        )

    def format_data(self):
        dev = self._factor_load("dev.tsv")
        # Not included: They were invalidated for a reason
        # invalidated = self._factor_load("invalidated.tsv")
        other = self._factor_load("other.tsv")
        test = self._factor_load("test.tsv")
        train = self._factor_load("train.tsv")
        validated = self._factor_load("validated.tsv")

        self.full_dataframe = self._merge_dataframes_rec([
            dev, other, test, train, validated
        ], 0)
        return self.full_dataframe
```

Listing 9: A fragment of the datasets.py script featuring the CommonVoiceES class for the adapter

```
Note: Some data fields have been trimmed for the sake of
   readability (client_id, i.e. 553951...b9)

Columns:
client_id path  sentence  up_votes  down_votes  age gender  accent

-> dev.tsv
[...]
553951...b9 common_voice_es_19746113.mp3  Su auge se dio con el
   cambio de siglo.  2 0
553951...b9 common_voice_es_19746114.mp3  Es originario del oeste
   de África tropical y de Borneo.  2 1
553951...b9 common_voice_es_19746115.mp3  Actualmente milita en el
   club Oriente Petrolero de la Primera Division de Bolivia.  2 0

[...]

-> invalidated.tsv
[...]
5d616f...bb common_voice_es_18306564.mp3  La plaza estaba muy
   concurrida  0 3 thirties  male  nortepeninsular
5d616f...bb common_voice_es_18306845.mp3  en el fin del mundo es
   como regalarte la vida . 0 2 thirties  male  nortepeninsular
5d616f...bb common_voice_es_18306846.mp3  ¡ Oye ! ¡ áest enferma !
   ¡ no , no lo áest ! 0 2 thirties  male  nortepeninsular
[...]

-> other.tsv
[...]
3cc1ab...1b common_voice_es_19592285.mp3  Esta historia transcurre
   en la ciudad portuaria de Libra. 0 0 twenties  male  rioplatense
3cc1ab...1b common_voice_es_19592327.mp3  Ninguno de los pasajeros
   ósobrevivi al accidente.  1 0 twenties  male  rioplatense
3cc1ab...1b common_voice_es_19594287.mp3  En todas las versiones,
   Sugimura muere casi al final de la historia.  1 0 sixties male
   surpeninsular
[...]

-> test.tsv
[...]
0003b9...5d common_voice_es_19698530.mp3  Habita en aguas poco
   profundas y rocosas. 2 1 thirties  male  mexicano
0003b9...5d common_voice_es_19987333.mp3  Opera principalmente
   vuelos de cabotaje y regionales de carga.  2 1
0003b9...5d common_voice_es_19691402.mp3  Para visitar contactar
   primero con la ódireccin. 2 0
[...]

-> train.tsv
[...]
434506...c2 common_voice_es_19742144.mp3  Tras su lanzamiento ha
   recibido positivas ñreseas por parte de la ícrtica especializada
   .  2 1 thirties  male  chileno
```

```
434506...c2 common_voice_es_19742146.mp3  Las hojas se secan a la
    sombra, en un lugar aireado.  2 1 thirties  male  chileno
434506...c2 common_voice_es_19742323.mp3  Por este motivo no pudo
    integrar la óseleccin de su ípas. 2 0 thirties  male  chileno
[...]

-> validated.tsv
[...]
0003b9...5d common_voice_es_19698530.mp3  Habita en aguas poco
    profundas y rocosas. 2 1 thirties  male  mexicano
009891...11 common_voice_es_19987333.mp3  Opera principalmente
    vuelos de cabotaje y regionales de carga.  2 1
00b0a5...aa common_voice_es_19691402.mp3  Para visitar contactar
    primero con la ódireccin. 2 0
[...]
```

Listing 10: Small sample of the rows contained in the input data index files of the CommonVoiceES dataset

2.2.1.2.8 LibriVoxES1 Adapter class This dataset was a bit complex to interface, it contains speech from male and female separated but also a part with mixed speech. The female part corresponds only to a single person speaking (so-called Karen Savage). The column names need to be overridden, as well as the filenames for they were .csv files. The separator used in these files was the pipe character ’|’. It contains several folders for each speaker, reading books. For brevity, only some of them are included in the example below.

```
¿¡ºª°’
class LibriVoxES1(Base):
    def __init__(self):
        super().__init__("librivox_es_1")
        self.COL_NAMES = ["path", "sentence", "rep_sentence"]
        self.CSV_FILENAME = "metadata.csv"
        self.AUDIO_FOLDER_NAME = "wavs"

    def _find_csv_folders(self):
        csv_folder_paths = []
        for root, dirs, files in walk(self.DATASET_FOLDER, topdown=
    True):
            for name in dirs:
                if name.split("/")[-1] == self.AUDIO_FOLDER_NAME:
                    full_path = path.join(root, name).split("/")
                    path_from_clips = "/".join(
                        full_path[full_path.index("clips") + 1 :
    -1]
                    )
                    csv_folder_paths.append(path_from_clips)
        return csv_folder_paths

    def format_data(self):
        csv_folders = self._find_csv_folders()
        dataframes = []
        for csv_folder_path in csv_folders:
            try:
                dataframes.append(self._load_from_tsv(
                    path.join(
                        self.DATASET_FOLDER, "clips",
                        csv_folder_path, self.CSV_FILENAME
                    ),
                    path.join(csv_folder_path, self.AUDIO_FOLDER_NAME) + "/",
                    separator="|",
                    path_suffix=".wav"
                ))
            except:
                print("[ERROR] -- There was a problem with dataset on {}"
                    .format(csv_folder_path), end=" --> ")
                print("Ommiting dataset")

        self.full_dataframe = self._merge_dataframes_rec(dataframes, 0)
        return self.full_dataframe
```

Listing 11: A fragment of the datasets.py script featuring the LibriVoxES1 class for the adapter

```
-> karen_savage/angelina/metadata.csv
[...]
angelina_00_delgado_f000001|CAPITULO UNO.|CAPITULO UNO.
angelina_00_delgado_f000002|RAFAEL DELGADO Y SU NOVELA ANGELINA.|RAFAEL DELGADO Y SU NOVELA ANGELINA.
angelina_00_delgado_f000003|Con este libro obtuvo el gran novelista mexicano el áms sonado éxito;|Con este libro obtuvo el gran novelista mexicano el áms sonado éxito;
[...]

-> male/tux/el_19_de_marzo_y_el_2_de_nayo/metadata.csv
[...]
el19demarzoyel2demayo_01_perezgaldos_f000001|íCaptulo Primero.|íCaptulo Primero.
el19demarzoyel2demayo_01_perezgaldos_f000002|El Diecinueve de Marzo y el Dos de Mayo de Benito éPrez óGalds.|El Diecinueve de Marzo y el Dos de Mayo de Benito éPrez óGalds.
el19demarzoyel2demayo_01_perezgaldos_f000003|En Marzo de mil ochocientos ocho, y cuando íhaban transcurrido cuatro meses desde que éempec a trabajar en el oficio de cajista, ya ícompona con mediana destreza, y ganaba tres reales por ciento de ílneas en la imprenta del Diario de Madrid.|En Marzo de mil ochocientos ocho, y cuando íhaban transcurrido cuatro meses desde que éempec a trabajar en el oficio de cajista, ya ícompona con mediana destreza, y ganaba tres reales por ciento de ílneas en la imprenta del Diario de Madrid.
[...]

-> male/victor_villarraza/cuentos_clasicos_del_norte/metadata.csv
[...]
cuentosclasicosprimera_00_poe_f000001|ÓINTRODUCCIN.|ÓINTRODUCCIN.
cuentosclasicosprimera_00_poe_f000002|Los cuatro escritores cuyas obras áestn representadas en esta ócoleccin son idealistas en uno u otro sentido.|Los cuatro escritores cuyas obras áestn representadas en esta ócoleccin son idealistas en uno u otro sentido.
cuentosclasicosprimera_00_poe_f000003|La literatura áclsica de los Estados Unidos no tiene realistas|La literatura áclsica de los Estados Unidos no tiene realistas
```

```
[...]

-> mix/la_condenada/metadata.csv
[...]
lacondenada_01_blasco_f000001|LA CONDENADA.|LA CONDENADA.
lacondenada_01_blasco_f000002|Catorce meses llevaba Rafael en la
    estrecha celda.|Catorce meses llevaba Rafael en la estrecha
    celda.
lacondenada_01_blasco_f000003|íTena por mundo aquellas cuatro
    paredes, de un triste blanco de hueso, cuyas grietas y
    desconchaduras se ísaba de memoria;|íTena por mundo aquellas
    cuatro paredes, de un triste blanco de hueso, cuyas grietas y
    desconchaduras se ísaba de memoria;
[...]
```

Listing 12: Small sample of the rows contained in the input data index files of the LibriVoxES1 dataset

2.2.1.2.9 LibriVoxES2 Adapter class Taken from the Librivox source, this dataset contains a total of 17 speakers. The data is all referenced from a single file. Only in this case, we needed to override the names and order of the columns and also the name of the file. In this case, we found a .csv file.

```
¿¡ºª°’
class LibriVoxES2(Base):
    def __init__(self):
        super().__init__("librivox_es_2")
        self.COL_NAMES = ["path", "wav_filesize", "sentence"]
        self.CSV_FILENAME = "files.csv"

    def format_data(self):
        self.full_dataframe = self._load_from_tsv(
            path.join(self.DATASET_FOLDER, self.CSV_FILENAME),
            "",
            separator=",",
            skip_rows=1,
            path_suffix=""
        )
        return self.full_dataframe
```

Listing 13: A fragment of the datasets.py script featuring the LibriVoxES2 class for the adapter

```
Columns:
wav_filename,wav_filesize,transcript

-> files.csv
[...]
audios/4da6b70e-0108-4f75-80ae-3d71f1dd2c2b.wav,219064,y íaqu en
    dos palotadas hemos encontrado robustas columnas donde apoyar la
     grandiosa áfbrica de su alcurnia
audios/8c2ab30b-0fd4-41c3-9724-3b15f2ee2c27.wav,271910,cuando los
    consejeros escucharon aquello quedaron estremecidos y se dijeron
     dios ha prohibido que padres se casen con sus hijas
audios/ca73c951-c62a-41fe-a953-9871514151f2.wav,64520,su mujer con
    la cara entre las manos
```

```
[...]
```

Listing 14: Small sample of the rows contained in the input data index files of the LibriVoxES2 dataset

2.2.1.2.10 Peruano Adapter class The Peruvian accent is represented in this dataset with a total of two files. Male and female speakers, one for each file.

```
¿¡ºªº’
class Peruano(Base):
    def __init__(self):
        super().__init__("peruanos")

    def format_data(self):
        line_index_female = self._load_from_tsv(
            path.join(self.DATASET_FOLDER, "line_index_female.tsv")
   ,
            "es_pe_female" + path.sep
        )
        line_index_male = self._load_from_tsv(
            path.join(self.DATASET_FOLDER, "line_index_male.tsv"),
            "es_pe_male" + path.sep
        )
        self.full_dataframe = self._merge_dataframes_rec([
            line_index_female, line_index_male
        ], 0)
        return self.full_dataframe
```

Listing 15: A fragment of the datasets.py script featuring the Peruano class for the adapter

```
-> line_index_female.tsv
[...]
pef_02436_00866988356 En el canal cien hay programas para ñnios
    entre cinco y nueve ñaos.
pef_09334_01927629957 La tienda de jabones y perfumes finos tienen
    regalos para el ída de las madres.
pef_09697_01394303270 ¿Sabe usted ácuntos estadios de úftbol
    profesional hay áac?
[...]

-> line_index_male.tsv
[...]
pem_01208_01446525215 Tocar el óxilfono es mi hobby favorito
pem_01523_01928013518 Inmediatamente te íenvo toda la óinformacin a
    tu correo
pem_03034_01073545033 ¿Es la violencia innata al ser humano?
[...]
```

Listing 16: Small sample of the rows contained in the input data index files of the Peruano dataset

2.2.1.2.11 PuertoRico Adapter class Another Spanish accent dataset, only in this case there is only a single file referencing audios from female speakers.

```
¿¡ºªº’
class PuertoRico(Base):
    def __init__(self):
        super().__init__("puerto_rico")

    def format_data(self):
        self.full_dataframe = self._load_from_tsv(
            path.join(self.DATASET_FOLDER, "line_index_female.tsv")
   ,
            "es_pr_female" + path.sep
        )
        return self.full_dataframe
```

Listing 17: A fragment of the datasets.py script featuring the PuertoRico class for the adapter

```
-> line_index_female.tsv
[...]
prf_06136_00202343619 Para lo que tu haces necesitas una
   computadora Apple
prf_06136_00990637626 El Llano de Llamas es un áclsico de Juan
   Rulfo
prf_04310_02013160949 Ahora mismo hay una oferta de mil trescientos
    ódlares americanos
[...]
```

Listing 18: Small sample of the rows contained in the input data index files of the PuertoRico dataset

2.2.1.2.12 TEDxES Adapter class This dataset comes from transcribed TEDx independent events. It includes a file named 'TEDx_Spanish.transcription' that has the sentence label and the filename of the audio it corresponds to separated by blank spaces. This one needed to be reconstructed into the needed format.

```
¿¡ºª°,
class TEDxES(Base):
    def __init__(self):
        super().__init__("tedx_es")
        self.PATHS_FILE_PATH = "files/TEDx_Spanish.paths"
        self.TRANSCRIPTIONS_FILE_PATH = "files/TEDx_Spanish.
   transcription"

    def _compose_data(self):
        paths = None
        transcriptions = None
        with open(
            path.join(self.DATASET_FOLDER, self.PATHS_FILE_PATH), "
   rt"
            ) as paths_file:
            paths = paths_file.readlines()
        with open(
            path.join(self.DATASET_FOLDER, self.
   TRANSCRIPTIONS_FILE_PATH), "rt"
            ) as transcriptions_file:
            transcriptions = transcriptions_file.readlines()

        dataframe = {"client_id": [], "path": [], "sentence": []}
        for p_line, t_line in zip(paths, transcriptions):
            relative_path = "/".join(p_line.split("/")[1:]).strip()
            transcript = " ".join(t_line.split(" ")[:-1]).strip()
            dataframe["client_id"].append(str(uuid4()))
            dataframe["path"].append(path.join(self.DATASET_FOLDER, "clips",relative_path))
            dataframe["sentence"].append(transcript)
        return pd.DataFrame(dataframe)

    def format_data(self):
        self.full_dataframe = self._compose_data()
        return self.full_dataframe
```

Listing 19: A fragment of the datasets.py script featuring the TEDxES class for the adapter

```
-> TEDx_Spanish.paths
[...]
./speech/TEDX_F_001_SPA_0001.wav
./speech/TEDX_F_001_SPA_0002.wav
./speech/TEDX_F_001_SPA_0003.wav
[...]

-> TEDx_Spanish.transcription
[...]
y eso se para ím se se puede reducir en équ en un des pertar de la conciencia humana gracias TEDX_F_001_SPA_0001
bueno e les voy a platicar una una historia y ella es yamila TEDX_F_001_SPA_0002
si les ponen a una mujer ías enfrente y les dicen éñensale a leer y a escribir TEDX_F_001_SPA_0003
[...]
```

Listing 20: Small sample of the rows contained in the input data index files of the TEDxES dataset

2.2.1.2.13 TUXES Adapter class This dataset contained two sub-datasets each with a 'metadata.csv' file that referenced the audio files using a pipe character as the separator '|'.

```
¿¡ºª°’
class TUXES(Base):
    def __init__(self):
        super().__init__("tux_es")
        self.COL_NAMES = ["path", "sentence", "rep_sentence"]
        self.CSV_FILENAME = "metadata.csv"
        self.AUDIO_FOLDER = "wavs"

    def format_data(self):
        other = self._load_from_tsv(
            path.join(self.DATASET_FOLDER, "clips", "other", self.CSV_FILENAME),
            path.join("other", self.AUDIO_FOLDER) + path.sep,
            separator="|"
        )
        valid = self._load_from_tsv(
            path.join(self.DATASET_FOLDER, "clips", "valid", self.CSV_FILENAME),
            path.join("valid", self.AUDIO_FOLDER) + path.sep,
            separator="|"
        )
        self.full_dataframe = self._merge_dataframes_rec([
            other, valid
        ], 0)
        return self.full_dataframe
```

Listing 21: A fragment of the datasets.py script featuring the TUXES class for the adapter

```
-> clips/other/metadata.csv
[...]
0|Mirad que os afusilamos si no ídecs la verdad|mirad que os
    afusilamos si no ídecs la verdad
1|éEsprese un poco ícarsimo maestro y ácapelln|éesprese un poco
    ícarsimo maestro y ácapelln
2|Por si algo pudiera valer, íhaba entregado al comendador la
    correspondencia de entrambos personajes, en que su trama estaba
    de manifiesto, pero no óconsigui por esto dar treguas a su pesar
    |por si algo pudiera valer, íhaba entregado al comendador la
    correspondencia de entrambos personajes, en que su trama estaba
    de manifiesto, pero no óconsigui por esto dar treguas a su pesar
[...]

-> clips/valid/metadata.csv
[...]
0|no soy el Pedro Hillo de antes, de tantos ñaos ípacficos y
    obscuros dentro de la paz sacerdotal|no soy el pedro hillo de
    antes, de tantos ñaos ípacficos y obscuros dentro de la paz
    sacerdotal
1|Y otros muchos sujetos muy dignos de hacer ómencin de ellos|y
    otros muchos sujetos muy dignos de hacer ómencin de ellos
2|Y esperaban que el jefe lo diera todo hecho|y esperaban que el
    jefe lo diera todo hecho
[...]
```

Listing 22: Small sample of the rows contained in the input data index files of the TUXES dataset

2.2.1.2.14 Venezolano Adapter class This dataset includes audio files from speakers with Venezuelan accents. In this case, we also have two files, one for male and one for female speakers.

```
¿¡ºª°’
class Venezolano(Base):
    def __init__(self):
        super().__init__("venezolanos")

    def format_data(self):
        line_index_female = self._load_from_tsv(
            path.join(self.DATASET_FOLDER, "line_index_female.tsv")
    ,
            "es_ve_female" + path.sep
        )
        line_index_male = self._load_from_tsv(
            path.join(self.DATASET_FOLDER, "line_index_male.tsv"),
            "es_ve_male" + path.sep
        )
        self.full_dataframe = self._merge_dataframes_rec([
            line_index_female, line_index_male
        ], 0)
        return self.full_dataframe
```

Listing 23: A fragment of the datasets.py script featuring the Venezolano class for the adapter

```
-> line_index_female.tsv
[...]
vef_02484_01513680092 ¿Ya sabes ácunto van a_letter costar las
    entradas?
vef_06136_01246918517 ¿Usted áest interesado en las olimpiadas
    o_letter en en el mundial de gimnasia que menciono édespus?
vef_07508_01579674640 ¿Puedes ayudarme a meditar?
[...]

-> line_index_male.tsv
[...]
vem_05223_00896110924 Los corazones de pollo son una delicia
vem_04310_01196944169 Es un plato muy nutritivo
vem_02484_00854567505 En este momento estoy enviando a sus mails
    unos links de unas meditaciones en You Tube
[...]
```

Listing 24: Small sample of the rows contained in the input data index files of the Venezolano dataset

2.2.1.2.15 WPHSSES Adapter class This dataset was by far the most difficult to process. There were several reasons as to why it was left out of the training samples:

- Too complex folder structure,
- The input files were scattered into questions, answers, prompts and recordings. This way it was really difficult to grasp how the dataset was built,
- The files contained missing characters lost by the file encoding. We made an attempt at recovering them using a Spanish dictionary, string manipulation and the fuzzywuzzy module to look for good word replacements, but we were not successful,

In the end, the dataset wasn't big enough for us to invest the time needed to make the interface work, so it was removed from the main script.

```
¿¡ºª°’
class WPHSSES(Base):
    # Useless (?)
    def __init__(self):
        super().__init__("wphss_es")

    def _clean_data(self):
        with open(path.join(self.DATASET_FOLDER, "questions.txt"))
   as filecontents:
            lines = filecontents.readlines()
            # Recover the initial question mark
            lines = [
                line.strip().replace("?", "¿", 1)
                for line
                in filter(
                    lambda line: len(line.strip().replace("?", "¿",
     1)) > 0, lines
```

```
                )
            ]
            # Get the individual tokens that contain missing characters
            tokens = []
            temp = [line.split(".")[-1].strip()[1:-1] for line in lines]
            for line in temp:
                split_lines = line.split(" ")
                for token in split_lines:
                    if "?" in token:
                        tokens.append(token)
            tokens = list(dict.fromkeys(tokens))
            # Load spanish dictionary to look for comparison with missing words
            spanish_tokens = None
            with open("/stt/recogida-audios/audio-processing/espanol.txt", "rt") as dictionary:
                spanish_tokens = dictionary.readlines()
                spanish_tokens = [
                    spanish_token.strip()
                    for spanish_token
                    in spanish_tokens
                    if any([
                        diacritic in spanish_token.strip()
                        for diacritic
                        in ["á", "é", "í", "ó", "ú", "ñ"]
                    ])
                ]
            # Look for a good replacement using fuzzywuzzy
            print((
                "[INFO] -[ (1/4) ]- "
                "Looking for best matches on missing diacritic characters"
            ))
            pbar = tqdm(tokens)
            extractions = []
            for token in pbar:
                pbar.set_description(token)
                extractions.append(process.extractOne(token, spanish_tokens))
            # Replace damaged tokens with best approximation
            print((
                "[INFO] -[ (1/4) ]- "
                "Replacing missing diacritic characters found by fuzzywuzzy"
            ))
            pbar2 = tqdm(zip(tokens, extractions))
            for token, replacement in pbar2:
                pbar2.set_description(token)
                for line in lines:
                    if token in line:
                        print(line)
                        line = line.replace(token, replacement[0])
                        print(line)

    def format_data(self):
        self._clean_data()
        return pd.DataFrame()
```

Listing 25: A fragment of the datasets.py script featuring the WPHSSES class for the adapter

```
Note: These are scattered files. Not included in the final training
    dataset

-> heroico-recordings.txt
[...]
1 iturbide se auto ónombr ígeneralsimo de mar y tierra
2 alarmado chile le ópidi al úper que declarara su neutralidad
3 anastasio somoza se órefugi primero en los estados unidos y
   édespus en paraguay
[...]

-> questions.txt
[...]
1. ?Cu?ndo fue la ?ltima vez que usted vio esta persona?
2. ?En qu? lugares se ha encontrado usted con esta persona?
3. ?Usted conoce bien a esta persona?
[...]
20. ?Cu?l es su nacionalidad?
21. ?Cu?l es su religi?n?
22. ?A qu? partido pol?tico pertenece?
[...]

-> usma-prompts.txt
[...]
s1  vivo en una casa
s2  ódnde vives tu
s3  eres de los estados unidos verdad
[...]

-> heroico-answers.txt
[...]
100/10  no ella no tiene barba ni bigote
100/11  de ni un color
100/12  negro
[...]
```

Listing 26: Small sample of the rows contained in the input data index files of the WPHSSES dataset

2.2.2 Splitting the dataset

The next step of the preprocessing is related to generating partitions of the data to train the model. We need data for three training phases: training, validation and testing.

2.2.2.1 The run.py script This partitioning was done using a script created by me for this purpose. Named as run.py and included under 'final-degree-project/code_repos/recogida-audios/dataset-processing', this script manages all the partitioning of the data and has more functionalities. Here is a breakdown of the parameters it accepts:

- **–merge-datasets:** Indicates whether or not the datasets should be merged into one before splitting,

- **–shuffle:** Makes the program shuffle the data before any splitting or export,
- **–no-split (MUTEX with –data-split):** Makes the program leave the data unsplitted,
- **–data-split (MUTEX with –no-split):** Expects a set of three integer values used to split data into three subsets. The values in the tuple, separated by blank space follow this correspondence: training, validation, testing.

Moreover, the procedure applied by this script is described as so, by the UML Activity diagram displayed in figure 11.

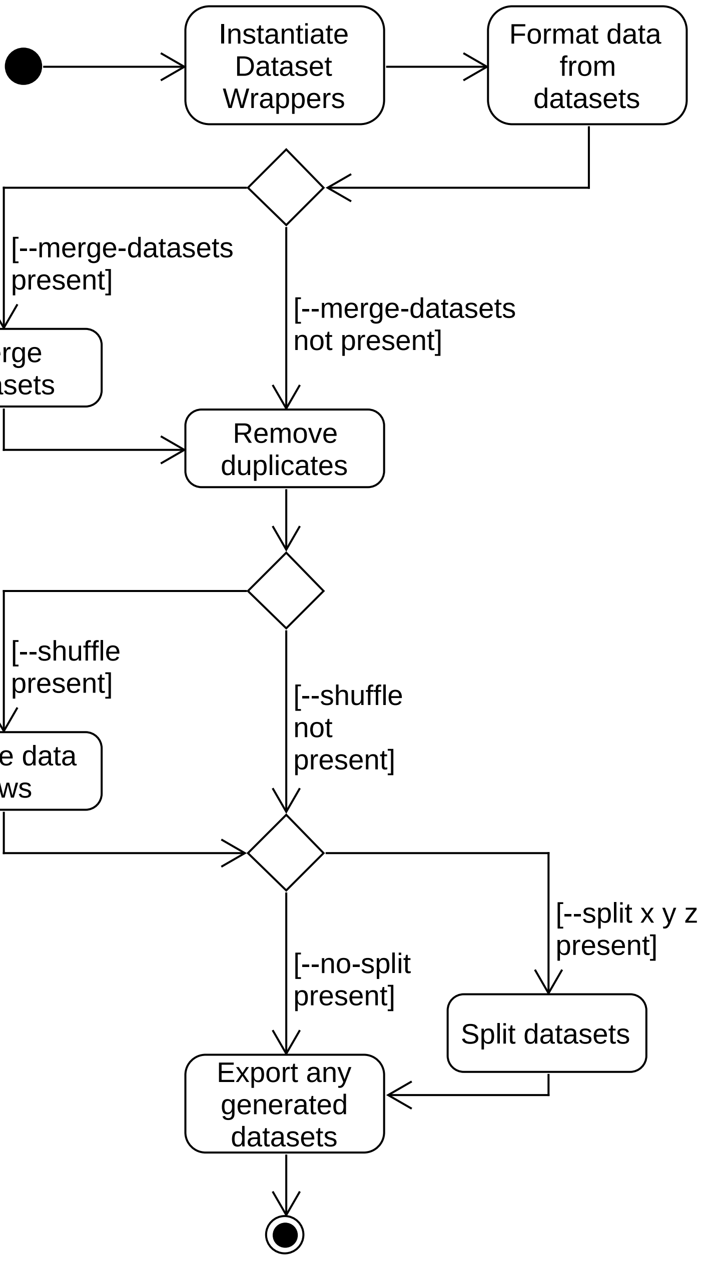


Figure 11: A UML Activity Diagram laying out the procedure followed by the run.py script used to split the training data before actually training the model

2.2.2.2 Executing the script

This script allows you to generate different sets of training files. You could use –merge-datasets if you have plenty of datasets but only want to use a file with all the samples. Also, using the –shuffle flag is recommended if –merge-datasets and –split are present to make a homogeneous split of data. This way, it is highly probable that all datasets have a percentage of representation in all the data splits. Not using –shuffle is also acceptable, but output datasets will probably be heterogeneous containing a lot of samples from a dataset and none from others.

Taking advantage of this script's design, a series of approaches were explored, only eventually, the homogeneous shuffled merge was the best option:

For the final training session, we used the run.py script to create 3 data partitions. The command used was:

```
~/stt$ python recogida-audios/dataset-processing/run.py \
    --merge-datasets \
    --shuffle \
    --split 89 10 1
```

Listing 27: Bash command to run the Python script used to split the datasets before training

2.2.2.3 Why did we use 89 10 1?

Having a total of 395087 samples in the merged dataset, the chosen split is:

- 89% of training data with a total of 351651 samples,
- 10% of validation data with a total of 39481 samples,
- 1% of test data with a total of 3955 samples of data.

Let us explain these quantities:

- **Training data:** The training data is used as the basis for learning. It should be obvious to know that the more data used for training, the better. 89% is the maximum available data taking into account the other two partitions.
- **Validation data:** During this phase, the model will attempt to perform inference on a subset of samples to test its ability. We do not need a big set of samples but using too few could give biased results. As will do if we took samples from the training dataset. 10% of the full dataset represents enough samples for this phase.
- **Test data:** This phase is the final one after all the Training-Validation Epochs. At this time in training, the model is assessed to see how well it performs with never-seen data. Metrics used to evaluate the results are calculated. We only need a few samples to obtain the best, and worst results. For this reason, 1% of the dataset is representative enough.

2.2.3 Handling noise

In this section, we have included a couple of items that attempted to remove noise from the dataset in the form of corrupted or unusable samples and silence periods in the audio files.

2.2.3.1 Trying to remove silence from the training samples

Indeed, silence is part of the speech we produce every day, but it is not so great when used as input to the model.

It introduces extra size in the files, resulting in fewer samples per training batch, ending with larger training times.

This kind of preprocessing was not present in the final iteration of the training data. Some attempts were made to minimize noise in the datasets by removing as much silence as possible from the audio files. This proposal was quickly discontinued since the results were far from acceptable. Moreover, the process of researching an efficient and lossless way of doing this would take up a lot of time.

Below we can see a script featuring the final test at removing the noise from the audio files.

```python
from os import listdir
from os.path import isfile, join, dirname
import numpy as np
import os

from pyAudioAnalysis import audioSegmentation as aS
from pyAudioAnalysis import audioBasicIO
import scipy.io.wavfile as wavfile

from pydub import AudioSegment

import warnings
warnings.filterwarnings("ignore")

def _remove_silences_from_audio(inputFile, outputFile, smoothingWindow=1.5, weight=0.3, plot=False):
    if not isfile(inputFile):
        raise Exception("Input audio file not found!")
    # Read audio file to look for silence fragments
    [fs, x] = audioBasicIO.read_audio_file(inputFile)
    segmentLimits = aS.silence_removal(x, fs, 0.05, 0.05,
                                       smoothingWindow, weight, plot)
    # Export each non-silent fragment to a wav file
    fragment_filenames = []
    for i, s in enumerate(segmentLimits):
        strOut = "{0:s}_{1:.3f}-{2:.3f}.wav".format(outputFile[0:-4], s[0], s[1])
        fragment_filenames.append(strOut)
        if not os.path.exists(dirname(outputFile)):
            os.makedirs(dirname(outputFile))
        wavfile.write(strOut, fs, x[int(fs * s[0]):int(fs * s[1])])
    # Merge the individual non-silent fragments into a new wav file
    fragments = [AudioSegment.from_wav(fragment) for fragment in fragment_filenames]
    output_wav_file = AudioSegment.empty()
    for fragment in fragments:
        output_wav_file = output_wav_file + fragment
    output_wav_file.export(outputFile, format="wav")
    # Remove the intermediate fragment wav files, not necessary anymore
    for temp_file in fragment_filenames:
        os.remove(temp_file)


# For testing purposes only
if __name__ == "__main__":
    mypath = "/stt/recogida-audios/dataset-processing/tests"
    onlyfiles = [join(mypath, f) for f in listdir(mypath) if isfile
(join(mypath, f))]
    for filename in onlyfiles:
        _remove_silences_from_audio(
            filename,
            join(dirname(filename), "output", filename.split("/")
[-1])
        )
```

Listing 28: audio_processor.py located under 'final-degree-project/code_repos/recogida-audios/dataset-processing'. Tries to remove silences from audio files

This script is trying to use the pyAudioAnalysis python package (See pyAudioAnalysis). We tried to remove the silence parts from the audio files one by one by segmenting them, writing these parts to files and then merging them without the silences. Simply put, it did not work.

On the one hand, this approach was very promising and feasible had we found a straightforward way to solve this problem.

On the other hand, the output files were worse than before because the library used in the above script was cutting too much audio. It was removing the silence but also trimming important audio containing speech. Several smoothing window values were tested; none of them gave significant differences from the original file.

After many trials and errors, this approach to remove silence from the audio files was abandoned.

2.2.3.2 Removing faulty or damaged audios from the training dataset During the first executions of the engine, we had runtime errors due to some audios not being accessible, corrupted, not found or any other reason as to being unable to use a sample.

To overcome this issue during the preprocessing phase, the files were programmatically opened one by one to see if they were readable, accessible on their path and checked whether they were really audio files in the specified format. If something failed, such a file's path was added to a file to be later excluded from the training datasets.

2.2.4 Exporting the dataset to the CommonVoice format

To conclude this section, we now explain the last step before training the model with the processed data.

2.2.4.1 Requirements of the my_import_cv script The DeepSpeech engine requires the data to be in a specific format. The data index files need to follow the next requirements:

- Only one file is allowed for each phase of the training process: Training per se, validation, and testing,
- Data rows need to have the following columns:
    - **wav_filename:** in my case, the filesystem path leading to the file represented in the current row,
    - **wav_filesize:** the size of the target file in Bytes,
    - **transcript:** the transcript label corresponding to the speech included in the audio in 'path'.

- Audio files must be in .wav format,
- Audio files must be at most 10 seconds long. Made to create reasonable size training batches,
- Audio files must be sampled at 16kHz. This can be tweaked in the script but it also constraints the training procedure. The engine needs to know what sample rate the audios use. It assumes 16kHz.

The original version of the script was taken from the DeepSpeech repo. I have tweaked its functionality to my personal needs of directory input, output, etc. Rather than leaving it as an open script and not really coupled with anything else, I have coupled it to the filesystem since it was not going to change for the duration of the experiment.

I just modified a couple of directory routes and paths to fit my dataset processing steps of merging several into the CommonVoice format so that this script was helpful enough for me not to create it from scratch.

Broadly speaking, this script takes the audio downloaded from Common Voice for a certain language, in addition to the *.tsv files output by CorporaCreator, and the script formats the data and transcripts to be in a state usable by DeepSpeech.py

2.2.4.2 Running the script With this command, we execute the script to generate the final dataset:

```
~/stt$ python recogida-audios/dataset-processing/my_import_cv2.py \
    --filter_alphabet DeepSpeech/data/alphabet.txt
    data/default
```

Listing 29: Bash command used to run the Python script my_import_cv2.py to generate the training datasets

The script will look for any .tsv files in the directory passed as a positional parameter. In this case, we don't need to specify the directory of the audios since the path are already included in each of the data rows. Moreover, the script looks for just files with one of the following names: test.tsv, dev.tsv,train.tsv, validated.tsv, other.tsv or full.tsv. Once found, loads the rows and tries to convert the audio file found at the path included in the data row to .wav. Then writes the path, the sentence transcript and the filesize in the new row to include in a .csv file with the same name.

For the final iteration of the training files, a total of three separate .tsv files were generated: train.tsv for the training phase, test.tsv for the testing phase and dev.tsv for validation phase.

These are examples of the files before and after the script is run:

```
-> dev.tsv (Before)
[...]
client_id path  sentence
0 179c906f-6c75-482f-addc-f02a51365232  /home/danifinca/stt/raw/stt
   /cv_es/clips/common_voice_es_19133840.mp3  El sobrino y su
   familia prosperaron en la óregin de Moss Vale.
1 0de67af7-a87a-447c-a1f1-0f07a128151c  /home/danifinca/stt/raw/stt
   /cv_es/clips/common_voice_es_19743833.mp3  Se encuentra en las
   Islas Canarias en Gomera.
2 ad8774cd-fdd2-47f4-b7e8-18f4aa4e51a9  /home/danifinca/stt/raw/stt
   /cv_es/clips/common_voice_es_18474260.mp3  Pasaremos unos ídas
   en el balneario, Marta
[...]

```

```
-> dev.csv (After)
[...]
wav_filename,wav_filesize,transcript
../cv_es/clips/common_voice_es_19133840.wav,202796,el sobrino y su
    familia prosperaron en la óregin de moss vale
../cv_es/clips/common_voice_es_18801584.wav,99884,iremos juntos
    hacia lleida
../cv_es/clips/common_voice_es_18474260.wav,166700,pasaremos unos
    ídas en el balneario marta
[...]
```

Listing 30: Fragments of the data index dev.tsv file before and after the my_import_cv2.py was run on the training datasets

2.3 Hyperparameter setup

In this section, we will take a look at the concept of hyperparameters. Moreover, we will also present the use of a technique called 'grid search' used to decide which values are the best to assign to these hyperparameters before training the model.

2.3.1 Definition of hyperparameters

A Machine Learning model depends on a great number of values that determine its accuracy when performing inference. We have simple parameters like weights and biases, for instance, and hyperparameters that are external to the model and do not change during training. If the training checkpoints are to be used in the future to fine-tune an existing model, these should be recorded somewhere. Some of them may require keeping their value when loading training checkpoints.

These parameters change depending on the model topology, but the ones interesting for us in DeepSpeech are:

- **Geometry of the network (–n_hidden):** This value controls the width of the Neural Network layers. It is advisable to use values which are powers of 2,
- **Learning Rate (–learning_rate):** This value controls the pace (or magnitude) at which parameters from the network are modified using the error propagated through the layers,
- **Dropout (–dropout_rate):** This value controls the percentage of updates done by backpropagation through the feed-forward layers. This prevents early overfitting during training.

2.3.2 Grid search for hyperparameter values

This approach is based on combinations of hyperparameters values. It is used to find a suitable combination of values to train the model and obtain better results.

The chosen values for each of the hyperparameters described above are:

- **–n_hidden:** 1024 and 2048,
- **–learning_rate:** 10^{-2} and 10^{-4},
- **–dropout_rate:** 0.3, 0.4, 0.5 and 0.6.

We could have added more, but taking into account that, with each added value the number of combinations grows exponentially, it is better to test with a few of them. This set of values combined generates a total of 16 combinations with each combination needing three DeepSpeech executions (later explained in 2.6).

This script was used to generate a bash script that would train a new model with each combination of the values and store the results, model output, checkpoints, and all generated files.

```python
import itertools
import os

NEURONS = [1024, 2048]
LEARNING_RATE = [10**-2, 10**-4]
DROPOUT_RATE = [0.3, 0.4, 0.5, 0.6]

template_command = """python -u DeepSpeech.py \
    --train_files /data/default/train.csv \
    --test_files /data/default/test.csv \
    --dev_files /data/default/dev.csv \
    --train_batch_size 32 \
    --dev_batch_size 32 \
    --test_batch_size 32 \
    --learning_rate %f \
    --dropout_rate %f \
    --n_hidden %d \
    --epochs 125 \
    --export_dir /exports \
    --early_stop True \
    --es_epochs 5 \
    --checkpoint_dir /checkpoints \
    --summary_dir /summaries \
    "$@" | tee /summaries/session_%d_log.txt
"""

export_commands = """DIRNAME=$(date +"%F_%H.%M.%S")
mkdir /helpers/$DIRNAME
cp -R -v /checkpoints /helpers/$DIRNAME/
cp -R -v /summaries /helpers/$DIRNAME/
cp -R -v /exports /helpers/$DIRNAME/
"""

clean_up_commands = """rm -rf /checkpoints
rm -rf /summaries
rm -rf /exports
"""

bash_exec = []

for xs in itertools.product(NEURONS, LEARNING_RATE, DROPOUT_RATE):
    for divisor, iteration in zip([1, 10, 20], [1, 2, 3]):
        bash_exec.append(template_command % (xs[1] / divisor, xs
    [2], xs[0], iteration))
        bash_exec.append(export_commands)
        bash_exec.append("\n")
    bash_exec.append(clean_up_commands)
    bash_exec.append("\n")

with open(os.path.dirname(__file__) + "/run-grid-search.sh", "w")
   as bash_out:
    for line in bash_exec:
        bash_out.write(line)

print(bash_exec)
```

Listing 31: Python script used to generate a bash script with DeepSpeech invocations to apply grid search to the model's hyperparameters

The python script writes to a file the enormous list of combinations and training commands to be used in the training environment. All the output files are moved in-between training sessions to keep a history of the whole process.

2.3.2.1 Final values obtained from grid search I created another script to parse all the training folder contents generated from the execution of the grid search bash script. This script generated an output 'summary.json' located under 'final-degree-project/code_repos/recogida-audios/scripts/reports'. It contains a list of JSON objects, each one representing each folder generated for a training session (meaning a combination from the grid search in conjunction with a number for the iterations of training).

The JSON file also contains all test phrases for each of the folders, with the values obtained for the metrics and all the list of JSON objects is sorted ascendingly by the value of the loss function, also present in each object.

The final result obtained from the grid search algorithm was:

```
[
    [...]
    {
        "folder_path": "/data/gs-history/2020-09-18_15.51.53",
        "parameters": {
            "dropout_rate": 0.4,
            "epochs": 125,
            "learning_rate": 1e-05,
            "n_hidden": 2048
        },
        "log_file_data": {
            "filename": "session_2_log.txt",
            "file_length": 96892,
            "epochs_done": 7,
            "test_metrics": {
                "WER": 0.252777,
                "CER": 0.07204,
                "loss": 16.600187
            },
            "results": {
                "Best WER:": [
                    {
                        "metrics": {
                            "WER": 0.0,
                            "CER": 0.107143,
                            "loss": 18.997168
                        },
                        "source_wav_file": "file:///data/default
    /../cv_es/clips/common_voice_es_18473718.wav",
                        "actual_transcription": "'uno dos  tres
    cuatro  cinco'",
                        "result_transcription": "'uno dos tres
    cuatro cinco'"
                    },
                {
                    "metrics": {
                        "WER": 0.0,
                        "CER": 0.0,
                        "loss": 5.131769
                    },
                    "source_wav_file": "file:///data/default/../librivox_es_1/clips/male/tux/la_batalla_de_los_arapiles/wavs/arapiles_16_perezgaldos_f000118.wav",
                    "actual_transcription": "'la óproteccin de jean jean era desinteresada o significaba un nuevo peligro mayor que los anteriores'",
                    "result_transcription": "'la óproteccin de jean jean era desinteresada o significaba un nuevo peligro mayor que los anteriores'"
                },
                {
                    "metrics": {
                        "WER": 0.0,
                        "CER": 0.033333,
                        "loss": 4.34259
                    },
                    "source_wav_file": "file:///data/default/../librivox_es_2/clips/audios/8cfe5b8f-0e9f-4e24-a62d-1d713218e8a2.wav",
                    "actual_transcription": "'vete hija ya debe de ser tarde'",
                    "result_transcription": "'vete hija  ya debe de ser tarde'"
                },
                {
                    "metrics": {
                        "WER": 0.0,
                        "CER": 0.0,
                        "loss": 4.097859
                    },
                    "source_wav_file": "file:///data/default/../librivox_es_2/clips/audios/7a8a222b-5c82-40e2-bb2a-2a34e680ab47.wav",
                    "actual_transcription": "'a la cama un soplo'",
                    "result_transcription": "'a la cama un soplo'"
                },
                {
                    "metrics": {
                        "WER": 0.0,
                        "CER": 0.0,
                        "loss": 3.845566
                    },
                    "source_wav_file": "file:///data/default/../cv_es/clips/common_voice_es_19843048.wav",
                    "actual_transcription": "'las tres cadenas ñmontaosas de los montes de crimea áestn representadas en sebastopol'",
                    "result_transcription": "'las tres cadenas ñmontaosas de los montes de crimea áestn representadas en sebastopol'"
                }
            ],
            "Median WER:": [
                {
                    "metrics": {
                        "WER": 0.2,
                        "CER": 0.041667,
                        "loss": 2.415637
                    },
                    "source_wav_file": "file:///data/default
    /../librivox_es_2/clips/audios/f8bc417d-565f-4d91-a4b7-
    ffbba0f9bdea.wav",
                    "actual_transcription": "'cuando se óindign
     al ver'",
                    "result_transcription": "'cuando se óindin
    al ver'"
                },
                {
                    "metrics": {
                        "WER": 0.2,
                        "CER": 0.058824,
                        "loss": 2.314881
                    },
                    "source_wav_file": "file:///data/default
    /../argentinos/clips/es_ar_female/arf_02436_01375746332.wav",
                    "actual_transcription": "'étens alguna
    preferencia de precio'",
                    "result_transcription": "'étens alguna
    presperencia de precio'"
                },
                {
                    "metrics": {
                        "WER": 0.2,
                        "CER": 0.022727,
                        "loss": 2.163583
                    },
                    "source_wav_file": "file:///data/default
    /../cv_es/clips/common_voice_es_18357059.wav",
                    "actual_transcription": "'amor por éinters
    se acaba en un dos por tres'",
                    "result_transcription": "'amor por éinters
    se acaba en un dos portres'"
                },
                {
                    "metrics": {
                        "WER": 0.2,
                        "CER": 0.026316,
                        "loss": 2.159589
                    },
                    "source_wav_file": "file:///data/default
    /../librivox_es_2/clips/audios/9f35bb64-8aef-4912-8857-
    b111bd0506d1.wav",
                    "actual_transcription": "'su aspecto
    general resultaba repelente'",
                    "result_transcription": "'su especto
    general resultaba repelente'"
                },
                {
                    "metrics": {
                        "WER": 0.2,
                        "CER": 0.038462,
                        "loss": 2.151875
                    },
                    "source_wav_file": "file:///data/default
    /../chilenos/clips/es_cl_male/clm_03034_00459022978.wav",
                    "actual_transcription": "'aprovechе su ída
    al ámximo'",
                    "result_transcription": "'aproveche su ída
    el ámximo'"
                }
            ]
        }
    }
  },
  [...]
]
```

Listing 32: A small fragment of the JSON file containing the results from the grid search algorithm sorted by loss function value

Hence for the final training, the values used were:

- **Geometry of the network (–n_hidden):** 2048 neurons per layer,
- **Learning Rate (–learning_rate):** 10^{-4} that is divided by 10 for the second session and by 20 for the third session,
- **Dropout (–dropout_rate):** 0.4.

2.4 Language model

The language model provides an additional layer of precision because it predicts which words are more likely to follow each other based on a vocabulary. It contains two sub-components, a KenLM Language Model and a trie data structure containing all words in the vocabulary.

The DeepSpeech's documentation calls this device a *scorer*, and it is used during the inference phase to improve model output. Refer to 1.5.3 for a more theoretical explanation.

Such documentation [21] indicates a series of steps to follow to create a new scorer from scratch:

1. **Look for a corpus with enough size to contain roughly every word in the target vocabulary.** To accomplish this, I made a Google search for Spanish text data corpus and found some resources. Most of them ended up being of no use since these were not correctly formatted, incomplete or contained corrupt characters due to accented vowels being in the alphabet. The resource from which I took the Spanish data corpus with which I created the scorer was SBWCE[1].
2. **Ensure the corpus includes one sentence per line only with words from the vocabulary and with characters from the alphabet.** After a scan through the corpus and gathering information about the problems I could have while creating the scorer, I created a Python script that would transform the input raw corpus into

[1]Cristian Cardellino: Spanish Billion Words Corpus and Embeddings (August 2019), https://crscardellino.github.io/SBWCE/

the correct shape needed to create the scorer file. There were some things I had to change or remove:

- Punctuation symbols were removed,
- Digits included were also erased,
- And any other character that was not included in our alphabet made that entire sentence be removed. In this group, I include words that, due to the file encoding, were not recognized as vowel-accented characters.

To help manage the raw data file a little bit more (12GB text file was hard to manage as a whole), it was segmented into parts using Ubuntu's command-line utility 'split'. We used it to cut the file into several parts, each containing 2500000 lines.
The following Python script was the one used to process the compressed text file containing the raw corpus downloaded from the cited resource. And it was invoked using this command:

```
~/deepspeech-scorer$ python cleaner.py sbwce.clean.part*

```

Listing 33: Bash command used to execute the cleaner script for the Spanish data corpus input to the scorer file creation phase

```
from os import path, linesep
from sys import argv
from tqdm import tqdm
import re

BAD_SENTENCES_FILENAME = "rejected.txt"

parent_dir = path.dirname(path.abspath(__file__))
txt_files = [path.join(parent_dir, filename) for filename in
    sorted(argv[1:])]

if len(txt_files) < 1:
    print("[ERROR] - No files were provided")
    exit(0)

print(f"[INFO] - Found a total of {len(txt_files)} file(s)")
txt_files_progress_bar = tqdm(txt_files, desc="File: ")
for file_part_path in txt_files_progress_bar:
    txt_files_progress_bar.set_description(path.basename(
    file_part_path))
    input_file = file_part_path
    output_file = input_file.replace(".txt", ".out.txt")
    with open(path.join(parent_dir, BAD_SENTENCES_FILENAME), "w
    ") as eliminated_lines_it:
        eliminated_lines = 0
        with open(input_file, "r") as input_file_it, \
            open(output_file, "w") as output_file_it:
            appended_lines = 0
            for line in input_file_it:
                # Remove new line character
                line = line.strip()
                # RegExp to match numbers and other not needed
    characters
                useless_chars = re.compile(r"
    ¿¡[0-9.,:;?!+\-*/|()€[\]{}$%&#@<>=]")
                line = re.sub(useless_chars, "", line)
                # RegExp to match lines with a capital I instead of an l (error)
                wrong_I_instead_of_l = re.compile(r"\b\w+I\w+\b")
                is_wrong_sentence = wrong_I_instead_of_l.findall(line)
                # RegExp to check for foreign characters other than our alphabet
                alphabet_negated_re = re.compile(r"[^a-zA-áéíóúÁÉÍÓÚñüZ ]")
                matches = alphabet_negated_re.findall(line)
                if matches or is_wrong_sentence:
                    eliminated_lines += 1
                    eliminated_lines_it.write(f"{line}{linesep}")
                else:
                    appended_lines += 1
                    output_file_it.write(f"{line.upper()}{linesep}")

            print(f"[INFO] - Eliminated a total of {eliminated_lines} line(s)")
            print(f"[INFO] - Kept a total of {appended_lines} line(s)")
            print(f"[INFO] - Processed a total of {appended_lines + eliminated_lines} line(s)")

```

Listing 34: cleaner.py Python script located under 'final-degree-project/code_repos/recogida-audios/scripts'. This script cleans the input Spanish text corpus removing specific sentences from the text containing problems

The script left out of the 'final.txt' file those sentences marked as not valid by the RegExp included in the script. Here is a short section of the file to see the kind of filter we applied to the data:

```
Hay mucho papeIeo que hacer
Un Ioco frustrado anda sueIto
Creo que tienes úItimo ída itis
EI tipo no ítena un bate sino una boIsa de deporte IIena de armasí
Queras saber Io que IIevaba
Los chicos iguaI me organizan aIgo
Mientras út áests jugando a ípoIicas yo estoy en casa pIaneando tu retiro
Te odio pero no éTambin te jubiIas

[...]ó

Decisin BCE  el Bulgarian National Bank y la Banca ţăNaional a âRomniei deben transferir al BCE los importes que ñacompaan a sus nombres en el cuadro del íartculo  de dicha óDecisin

[...]Ã

As pues los productos esperados son tres Ã©homopolipptidos diferentes
Los enlaces triples aparecen muy raramente en Ã©biomolcuLas
Algunas Ãprotenas contienen Ãms de un grupo Ã©prosttico
```

```
El ciclo de el Ãcido Ãctrico tiene ocho pasos
Facilitar la Ã³informacin que el Parlamento solicite a el Gobierno
El acuerdo Ãser notificado a la Entidad deudora
Turkel Ã³critic este estudio porque a el parecer no Ãreproduca correctamente su Ã©mtodo Ã©teraputico
En el patio juega animadamente y corre con un grupo de Ã±nios

```

Listing 35: A couple of fragments from the file that recorded the sentences that did not make it into the final corpus to create the scorer file

3. **Generate the trie binary file ('lm.binary') using KenLM tools and DeepSpeech's python scripts.** Following the documentation, I cloned the KenLM repository and executed the Python script called 'generate_lm.py' included in the DeepSpeech repository under '/DeepSpeech/data/lm/'. The executed command is:

```
~/deepspeech_scorer$ python DeepSpeech/data/lm/generate_lm.py \
    --input_txt ../sbwce/sbwce.clean.txt.gz \
    --output_dir . \
    --top_k 500000 \
    --kenlm_bins kenlm/build/bin \
    --arpa_order 5 \
    --max_arpa_memory "85%" \
    --arpa_prune "0|0|1" \
    --binary_a_bits 255 \
    --binary_q_bits 8 \
    --binary_type trie \
        | tee -a generate_lm.py.log

```

Listing 36: Bash command used to execute the generation script of the lm.binary file needed to create the Spanish scorer file

This command generates the 'lm.binary' trie file and a 'vocab-500000.txt' file. The former represents the trie data structure mentioned above, and the latter contains the 500000 most used words in the input corpus .txt file. These are sorted from more to less common.

4. **Use the native client binary executables from the DeepSpeech repo to generate the scorer file.** These binaries are found under the releases tree of the GitHub repository. The project version I used for this work corresponds to git history tag v0.8.0. The necessary file for a 64-bit CPU, Windows OS is obtained from this link: native_client.amd64.cpu.win.tar.xz. Inside this compressed file, we find several files. For this step, we need the one called 'generate_scorer_package'. It is a console application built for Windows but depending on the compressed file used, this changes depending on the OS and CPU architecture. The command needed to create the .scorer file is the following:

```
C:\Users\danal>./generate_scorer_package ^
    --alphabet ../alphabet.txt ^
    --lm lm.binary ^
    --vocab vocab-500000.txt ^
    --package kenlm.scorer ^
    --default_alpha 0.931289039105002 ^
    --default_beta 1.1834137581510284

```

Listing 37: Bash command used to execute the native client binary to generate the scorer

package out of the lm.binary and vocab-500000.txt files

To this command, I have inputted the output from the previous step and added default values for alpha and 'beta'. These correspond to the language model equation presented in section 1.5.3. The floats included in the command are the ones supplied by DeepSpeech's docs. The next steps show how to obtain an appropriate pair for our scorer. Plus, 'package.scorer' will be the output name of our scorer file.

5. **Use the scorer with alpha and beta, and some test files to calculate an optimized pair of values.** *Note*[2]. To achieve this, we need to execute yet another Python script. This time it is located under the root of the repo '/DeepSpeech/', named 'lm_optimizer.py'. Following the documentation, this script should be executed using the set of checkpoints obtained from the training session of the model. The command would be invoked using this command:

```
~/DeepSpeech$ python lm_optimizer.py \
        --test_files speech_data/example_folder/test.csv \
        --checkpoint_dir deepspeech-data/checkpoints

```

Listing 38: Bash command used to invoke the Python script that would use the training environment to calculate appropriate parameters alpha and beta to optimize the scorer's performance

It is important to know that, in the case of executing this with a different NN architecture, we need to include the specific changes to such architecture in this command. For instance, if we changed the –n_hidden parameter for the training, we need to specify that change here. If not done, the checkpoints will not be loaded and the program will crash. This would be the normal output of the script as explained in [20]:

```
# lm_optimizer.py will create a new study
[I 2021-03-05 02:04:23,041] A new study created in memory with
   name: no-name-38c8e8cb-0cc2-4f53-af0e-7a7bd3bc5159

[...]

# It will then run testing and output a trial score.
[I 2021-03-02 12:48:15,336] Trial 0 finished with value: 1.0
   and parameters: {'lm_alpha': 1.0381777700987271, 'lm_beta':
    0.02094605391055826}. Best is trial 0 with value: 1.0.

[...]

# By default, lm_optimizer.py will run 6 trials, and identify
   the trial with the most optimal parameters.
[I 2021-03-02 17:50:00,662] Trial 6 finished with value: 1.0
   and parameters: {'lm_alpha': 3.1660260368070423, 'lm_beta':
    4.7438794403688735}. Best is trial 0 with value: 1.0.

```

Listing 39: Expected output from the script that guesses the best values for alpha and beta when optimizing the scorer's operation

[2]This step could not be replicated in my experiment environment because at the time this was done, the computing resources used for the training process were no longer at my disposal.

6. **Create a new scorer package with the optimized values.** Once these values are obtained, we need to execute 'generate_scorer_package' again to regenerate the scorer package, this time using the alpha and beta values obtained in the previous step. This way, we end up with an optimized scorer for our target language and vocabulary.

2.5 Experimental setup

At first, the system had a Proxmox host installed that served as a virtualization platform to create both Virtual Machines and Containers. It was really hard to set up docker with all the requirements. On top of this, the platform was not configured to be able to use any graphics cards (GPUs). This was a major issue since Tensorflow uses NVIDIA's GPUs to speed up the process of calculating massive tensor multiplications.

I tried to enable what is known as "PCI-Passthrough", a setting that makes the PCI devices and hardware available to the guests created on top of the Proxmox environment. I finally achieved it, but the performance was affected by the extra layer of abstraction between the model training engine and the host managing the GPU, so in the end, we prepared a new host machine with a Ubuntu Server - Linux OS.

Apart from having the drivers of the GPU up-to-date and docker installed, we needed to install special extensions to make docker work with the GPU. Here is a list of links to pages with documentation or files I needed to use to make the machine work:

- CUDA ToolKit Documentation
- CUDA ToolKit 10.0 Archive
- DockerHub - TensorFlow:1.15.2-py3-jupyter
- TensorFlow Docs - About Docker
- Docker Docs - Install Docker in Ubuntu
- GitHub Repo - NVIDIA Docker

2.5.1 Dockerfile for training

All the training processes took place inside Docker containers. This way, the training could be developed in a controlled environment, be repeatable and easy to configure.

The DeepSpeech repository contains a template makefile to generate base Dockerfiles for training. This saved a lot of time in creating it, figuring out the needed dependencies, and fixing any compatibility issue that might have arisen.

This is the Dockerfile that was generated by the template:

```
# Please refer to the TRAINING documentation, "Basic Dockerfile for training"

FROM tensorflow/tensorflow:1.15.2-gpu-py3
ENV DEBIAN_FRONTEND=noninteractive

ENV DEEPSPEECH_REPO=https://github.com/mozilla/DeepSpeech.git
ENV DEEPSPEECH_SHA=f56b07dab4542eecfb72e059079db6c2603cc0ee

RUN apt-get update && apt-get install -y --no-install-recommends \
        apt-utils \
        bash-completion \
        build-essential \
        cmake \
        curl \
        git \
        libboost-all-dev \
        libbz2-dev \
        locales \
        python3-venv \
        unzip \
        wget

# We need to remove it because it's breaking deepspeech install
    later with
# weird errors about setuptools
RUN apt-get purge -y python3-xdg

# Install dependencies for audio augmentation
RUN apt-get install -y --no-install-recommends libopus0 libsndfile1

# Try and free some space
RUN rm -rf /var/lib/apt/lists/*

WORKDIR /
RUN curl -s https://packagecloud.io/install/repositories/github/git
    -lfs/script.deb.sh | bash
RUN apt-get install git-lfs
RUN git lfs install
RUN git clone $DEEPSPEECH_REPO

WORKDIR /DeepSpeech
RUN git checkout $DEEPSPEECH_SHA

# Build CTC decoder first, to avoid clashes on incompatible
    versions upgrades
RUN cd native_client/ctcdecode && make NUM_PROCESSES=$(nproc)
    bindings
RUN pip3 install --upgrade native_client/ctcdecode/dist/*.whl

# Prepare deps
RUN pip3 install --upgrade pip==20.0.2 wheel==0.34.2 setuptools
    ==46.1.3

# Install DeepSpeech
#  - No need for the decoder since we did it earlier
#  - There is already correct TensorFlow GPU installed on the base
    image,
#    we don't want to break that
RUN DS_NODECODER=y DS_NOTENSORFLOW=y pip3 install --upgrade -e .

# Tool to convert output graph for inference
RUN python3 util/taskcluster.py --source tensorflow --branch r1.15
    \
        --artifact convert_graphdef_memmapped_format  --target .

# Build KenLM to generate new scorers
WORKDIR /DeepSpeech/native_client
RUN rm -rf kenlm && \
    git clone https://github.com/kpu/kenlm && \
    cd kenlm && \
    git checkout 87e85e66c99ceff1fab2500a7c60c01da7315eec && \
    mkdir -p build && \
    cd build && \
    cmake .. && \
    make -j $(nproc)

WORKDIR /DeepSpeech

EXPOSE 8080

COPY ./data/alphabet.txt /DeepSpeech/data/
COPY ./training/deepspeech_training/evaluate.py /DeepSpeech/
    training/deepspeech_training/evaluate.py
COPY ./training/deepspeech_training/train.py /DeepSpeech/training/
    deepspeech_training/train.py
COPY ./bin/run-ES-ds.sh /DeepSpeech/bin/
RUN chmod +x /DeepSpeech/bin/run-ES-ds.sh
```

Listing 40: Contents of the Dockerfile used to create containers in which we could train the model

In the previous file, we see that the container is setup with all the necessary dependencies and that the DeepSpeech repo is cloned and checked out to the concrete tag needed for this version of the engine. Moreover, this file also instructs the container to build the KenLM tools, the CTC decoder and Python-required tools to execute the engine.

To run the container, we needed to set up shared volumes for the training process to save output files such as log files or checkpoints to disk. These were the commands used to run docker when creating training containers:

```bash
#!/bin/bash

cd /home/danifinca/stt/DeepSpeech
sudo docker stop ds-gpu-training
sudo docker rm ds-gpu-training
sudo docker build -t ds-gpu-image -f Dockerfile.train .
sudo docker run -it -d --name ds-gpu-training \
    --gpus all \
    -p 0.0.0.0:8089:8080 \
    -v /home/danifinca/stt/data:/data \
    -v /home/danifinca/stt/checkpoints:/checkpoints \
    -v /home/danifinca/stt/summaries:/summaries \
    -v /home/danifinca/stt/history:/helpers \
    -v /home/danifinca/stt/exports:/exports \
    ds-gpu-image
sudo docker exec -it ds-gpu-training bash
```

Listing 41: Bash shell script used to run the docker containers created for training with DeepSpeech

2.6 Model training

2.6.1 Training parameters

Model training, and many other processes performed by the DeepSpeech engine, are managed through the command line using (mostly) the DeepSpeech.py script. This script, located under '/DeepSpeech/', can receive a lot of flags to configure its operation. In this section of its documentation, we can see the definition of all of these flags, a small text indicating help about each flag, and the semantic group to which they belong. See section FLAGS in [21].

For easy reference, here is a detailed explanation of each of the flags that are relevant to our case study:

- **–train_files:** A comma-separated list of paths indicating where to find the dataset index files used for training. If more than one is included, these will be merged and if none are supplied, the engine doesn't start training,
- **–test_files:** Works in the same way as –train_files, but this time applies for the data index files used for testing,
- **–dev_files:** Works in the same way as –train_files, but this time applies for the data index files used for validation,
- **–train_batch_size:** Indicates the number of samples to use per batch during the training phase. This value needs to be fine-tuned to adapt to the memory available in the GPUs. It is better kept at values close to powers of 2,
- **–dev_batch_size:** Indicates the number of samples to use per batch during the validation phase. This value needs to be fine-tuned to adapt to the memory available in the GPUs. It is better kept at values close to powers of 2,
- **–test_batch_size:** Indicates the number of samples to use per batch during the testing phase. This value needs to be fine-tuned to adapt to the memory available in the GPUs. It is better kept at values close to powers of 2,
- **–learning_rate:** A double value indicating the learning rate of the ADAM optimizer. That is, the magnitude by which the optimizer incorporates new error corrections into the parameters of the network,
- **–dropout_rate:** A double value indicating the dropout rate of the feedforward layers. This value can be different for each layer using flags: '–dropout_rateX' with X being a number between 2 and 6 included,
- **–n_hidden:** The number of neurons in the hidden layers of the network. Defines its width and its used at Network's initialization,
- **–epochs:** The number of times the data will be swept through to train the model following the pattern: Training-Validation*Epochs then Testing*1,
- **–export_dir:** The path to a directory where the output model is to be stored when training finishes,
- **–early_stop:** A boolean value indicating the engine whether early stop should be activated. This strategy stops the training process automatically after '–es_epochs' epochs if the increment of the loss function value during the Validation phase does not improve (after not changing for '–es_epochs' epochs),
- **–es_epochs:** The number of epochs to watch the Validation phase loss-function value for changes. If it does not improve during this time and '–early_stop' is enabled, training is stopped,
- **–checkpoint_dir:** The path of the directory where the training checkpoints should be extracted,
- **–summary_dir:** This directory is used by Tensorflow to store TensorBoard summary files.

These are the flags that have been used for the final training of the model.

2.6.2 Running training

2.6.2.1 The training script The following script was the one used to train the final model inside the Docker container created from the Dockerfile.train template provided by DeepSpeech's repo.

```
#!/bin/sh
set -xe
if [ ! -f DeepSpeech.py ]; then
    echo "Please make sure you run this from DeepSpeech's top level directory."
    exit 1
fi;

# Force only one visible device because we have a single-sample dataset
# and when trying to run on multiple devices (like GPUs), this will break
export CUDA_VISIBLE_DEVICES=0

python -u DeepSpeech.py \
  --train_files /data/default/train.csv \
  --test_files /data/default/test.csv \
  --dev_files /data/default/dev.csv \
  --train_batch_size 32 \
  --dev_batch_size 32 \
  --test_batch_size 32 \
  --learning_rate 0.0001 \
  --dropout_rate 0.60 \
  --n_hidden 2048 \
  --epochs 125 \
  --export_dir /exports \
  --early_stop True \
  --es_epochs 5 \
  --checkpoint_dir /checkpoints \
  --summary_dir /summaries \
  "$@" | tee /summaries/session_1_log.txt

DIRNAME=$(date +"%F_%H.%M.%S")
mkdir /helpers/$DIRNAME
cp -R /checkpoints /helpers/$DIRNAME/
cp -R /summaries /helpers/$DIRNAME/
cp -R /exports /helpers/$DIRNAME/

python -u DeepSpeech.py \
  --train_files /data/default/train.csv \
  --test_files /data/default/test.csv \
  --dev_files /data/default/dev.csv \
  --train_batch_size 32 \
  --dev_batch_size 32 \
  --test_batch_size 32 \
  --learning_rate 0.00001 \
  --dropout_rate 0.60 \
  --n_hidden 2048 \
  --epochs 100 \
  --export_dir /exports \
  --early_stop True \
  --es_epochs 5 \
```

```bash
      --checkpoint_dir /checkpoints \
      --summary_dir /summaries \
      "$@" | tee /summaries/session_2_log.txt

    DIRNAME=$(date +"%F_%H.%M.%S")
    mkdir /helpers/$DIRNAME
    cp -R /checkpoints /helpers/$DIRNAME/
    cp -R /summaries /helpers/$DIRNAME/
    cp -R /exports /helpers/$DIRNAME/

    python -u DeepSpeech.py \
      --train_files /data/default/train.csv \
      --test_files /data/default/test.csv \
      --dev_files /data/default/dev.csv \
      --train_batch_size 32 \
      --dev_batch_size 32 \
      --test_batch_size 32 \
      --learning_rate 0.000005 \
      --dropout_rate 0.60 \
      --n_hidden 2048 \
      --epochs 100 \
      --export_dir /exports \
      --early_stop True \
      --es_epochs 5 \
      --checkpoint_dir /checkpoints \
      --summary_dir /summaries \
      "$@" | tee /summaries/session_3_log.txt

    DIRNAME=$(date +"%F_%H.%M.%S")
    mkdir /helpers/$DIRNAME
    cp -R /checkpoints /helpers/$DIRNAME/
    cp -R /summaries /helpers/$DIRNAME/
    cp -R /exports /helpers/$DIRNAME/
```

Listing 42: The Bash script used to train the final STT model inside a Docker container created from DeepSpeech's repo

2.6.2.2 Why three training steps?

The above script includes three executions of the DeepSpeech.py script for training. This decision was made based on the official releases page of DeepSpeech's repo. The development team wrote a section about how they trained their release models [21].

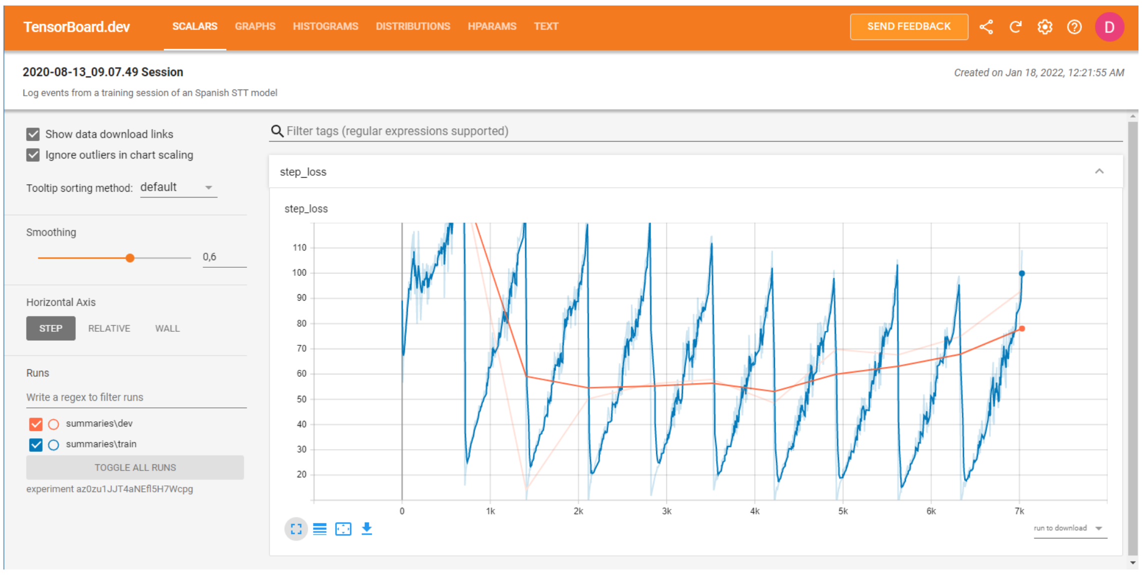
In that section, the team explains that they ran training over the training set three times. Each training session had the learning rate reduced as well as the training epochs. That made the engine fine-tune well their release model.

In our case, the training process followed their directives. In this case, we did not have the computing power of 8 GPUs as stated on the page, so we had to adapt the training parameters to our system's constraints. Adapting the batch size was already enough. Plus, early stopping was enabled with five stop epochs since our dataset was not big enough to train for 125 epochs. Most of our training sessions in the final model and during Grid Search execution took no longer than 25-30 epochs.

2.6.3 Training monitoring

This part of the training was crucial. It allowed us to know whether the model was improving its predictions over time. We used TensorBoard, which is a visualization toolkit complement to TensorFlow.

This tool allows us to deploy a Dashboard to monitor the training in real-time. It has this appearance:


2020-08-13_09.07.49 Session

Log events from a training session of an Spanish STT model

Created on Jan 18, 2022, 12:21:55 AM

Show data download links

Ignore outliers in chart scaling

Tooltip sorting method: default

Smoothing 0,6

Horizontal Axis: STEP

summaries\dev

summaries\train

experiment az0zu1JJT4aNEfI5H7Wcpg

Figure 12: An example of the dashboard shown in TensorBoard monitoring toolkit

Using this command below, we can host our copy of TensorBoard for monitoring purposes:

```
tensorboard --logdir /summaries/ --port <PORT> --host 0.0.0.0
```

Listing 43: Bash command to self-host a tensorboard dashboard to monitor the scalars folder of a model training sessions

2.6.3.1 Upload of the training sessions scalars

This tool can also be used in the cloud without any installation or private hosting. This service can be used by linking one's Google Account to upload the scalars.

With this command, we can upload a training session's data to the service and later consult the data in the same way as depicted in 12:

```
tensorboard dev upload --logdir 2020-08-19_05.36.16/ --name "
    2020-08-19_05.36.16 Session" --description "Log events from a
    training session of a Spanish STT model"
```

Listing 44: TensorBoard command used to upload the scalars of a training session to TensorBoard.dev

Here there is a couple of example training sessions to look at inside the monitor dashboard provided by TensorBoard.dev (associated with my Google Account):

- STT Session with almost 360k data points,
- Another STT Session this time with fewer data points; around 250k,

- Test run with 8k data points used to test the platform and the grid search algorithm.

Disclaimer: These URLs are publicly accessible. Anyone with the link can read the charts and download the information and data points hosted on the platform.

2.6.4 Model exports

The training engine allows the user to export the resulting weights and network configuration to a binary file usually referred to as "the model". It is just composed of numbers: weights and biases to configure the network when performing inference.

We have a couple of options for exporting this model:

- We can create a model.pb file. If the training command included the '–export_dir' flag, the engine will export the network's state to a .pb file,
- We can also export this same file in a format compatible with TensorFlow Lite, or tflite for short. In this case, we need to include the '–export_tflite' flag.

2.6.4.1 Transforming the exported model into an mmap-able model

The file exported from the training engine is loaded into memory when performing inference. This consumes a lot of memory and execution time. A way to avoid this, is to read data directly from the disk.

Following the project's documentation [21], we can use a tool included in the DeepSpeech's repo to upgrade this model file into a .pbmm file, allowing us to achieve faster loading times and inference.

Use these commands:

```
python3 util/taskcluster.py --source tensorflow --artifact
    convert_graphdef_memmapped_format --branch r1.15 --target .

[...]

convert_graphdef_memmapped_format --in_graph=output_graph.pb --
    out_graph=output_graph.pbmm
```

Listing 45: Bash commands to transform a .pb model file into a memory-mapped version .pbmm for better performance

2.7 Backend implementation prototype

I have created an API as a proof of concept to work with the trained models, send audios and test the accuracy. This backend is implemented in JavaScript using Node.js and Express.js.

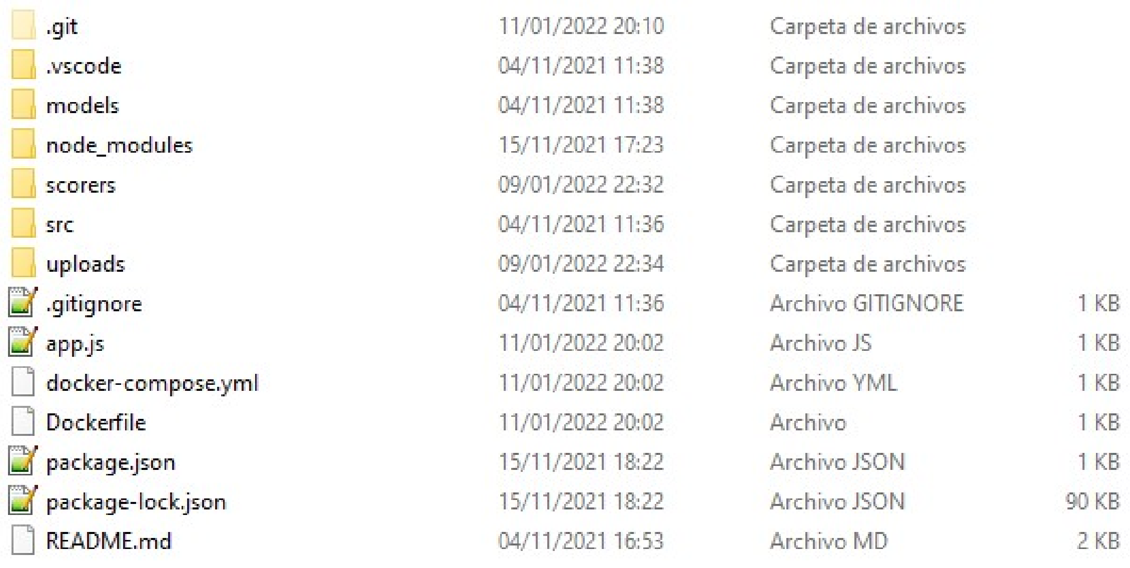
It only includes one endpoint to send audio files in .wav format for the model to transcribe. It may introduce some overhead in systems that may use this API, but the objective is to test if the models worked as intended. From then on, this backend could be upgraded and extended to feature more endpoints with greater capabilities.

2.7.1 Project file structure

The project folder's structure (see figure 13) is composed of several items needed for the app to function:

- **models/:** In this folder, you will find the different models exported from the training environment for the backend to use.
- **node_modules/:** This folder contains the dependencies of the project needed to execute the API. It is a really heavy object.
- **scorers/:** This folder contains the scorer files used as language models for the inference phase.
- **src/:** In this folder, we can find the JavaScript source files that implement the system.
- **uploads/:** This directory is created at runtime and stores the files sent by the clients in the server to be processed by the model.
- **app.js:** This file is the entry point of the application, contains many imports of the dependencies, creates the server which exposes the API and manages the different routes the API supports.
- **package.json:** This file contains information about the repo, the project, the creator, and the dependencies of the project.
- **package-lock.json:** This file is created whenever we perform 'npm install' on the system and contains the complete tree of dependencies apart from the ones included in package.json.

It is important to note that there needs to be at least one model file present. Otherwise, the API will crash at startup. The scorer is recommended to improve the model's transcriptions, but it is not mandatory.



.git	11/01/2022 20:10	Carpeta de archivos	
.vscode	04/11/2021 11:38	Carpeta de archivos	
models	04/11/2021 11:38	Carpeta de archivos	
node_modules	15/11/2021 17:23	Carpeta de archivos	
scorers	09/01/2022 22:32	Carpeta de archivos	
src	04/11/2021 11:36	Carpeta de archivos	
uploads	09/01/2022 22:34	Carpeta de archivos	
.gitignore	04/11/2021 11:36	Archivo GITIGNORE	1 KB
app.js	11/01/2022 20:02	Archivo JS	1 KB
docker-compose.yml	11/01/2022 20:02	Archivo YML	1 KB
Dockerfile	11/01/2022 20:02	Archivo	1 KB
package.json	15/11/2021 18:22	Archivo JSON	1 KB
package-lock.json	15/11/2021 18:22	Archivo JSON	90 KB
README.md	04/11/2021 16:53	Archivo MD	2 KB

Figure 13: An image of the contents of the backend project folder as described above

2.7.2 API documentation

UML diagrams are really useful to document the API and represent its architecture and functionality. We can find a couple of these diagrams below to help explain the backend structure.

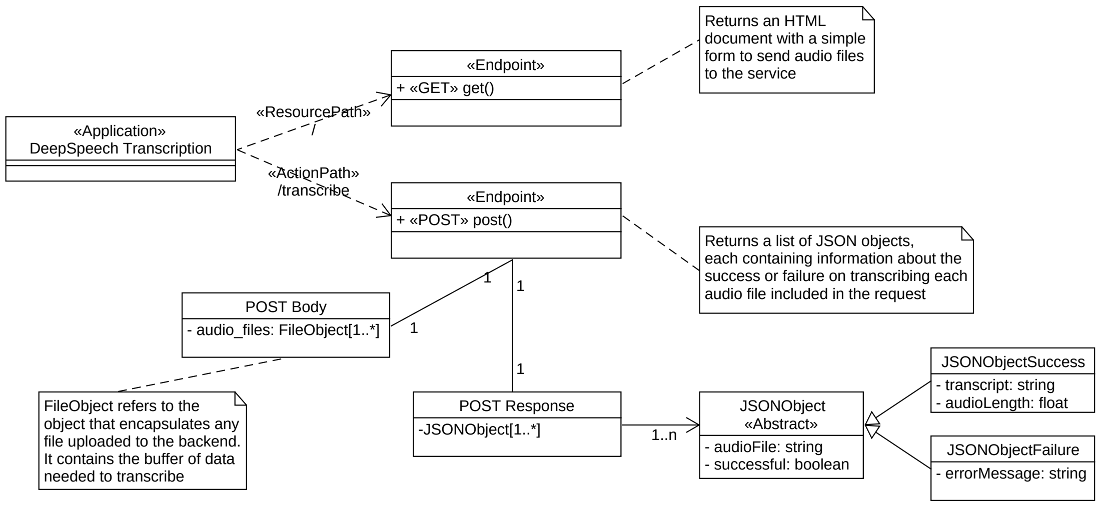

«Application»
DeepSpeech Transcription
«ResourcePath»
/
«ActionPath»
/transcribe
«Endpoint»
+ «GET» get()
Returns an HTML document with a simple form to send audio files to the service
«Endpoint»
+ «POST» post()
Returns a list of JSON objects, each containing information about the success or failure on transcribing each audio file included in the request
POST Body
- audio_files: FileObject[1..*]
1
1
1
1
FileObject refers to the object that encapsulates any file uploaded to the backend. It contains the buffer of data needed to transcribe
POST Response
-JSONObject[1..*]
1..n
JSONObject
«Abstract»
- audioFile: string
- successful: boolean
JSONObjectSuccess
- transcript: string
- audioLength: float
JSONObjectFailure
- errorMessage: string

Figure 14: Simple UML diagram to represent the path structure of the API with the available endpoints

In figure 14, we can see the two endpoints available for the clients. The root path '/' (using GET) only returns an HTML form that allows the upload of audio files. The other endpoint with path '/transcribe' (using POST) receives a list of audio files and returns a list of JSON objects containing information about each audio file. If the API is successful at transcribing the audio, the associated JSON object will contain the transcript and audio length. Otherwise, an error message is received.

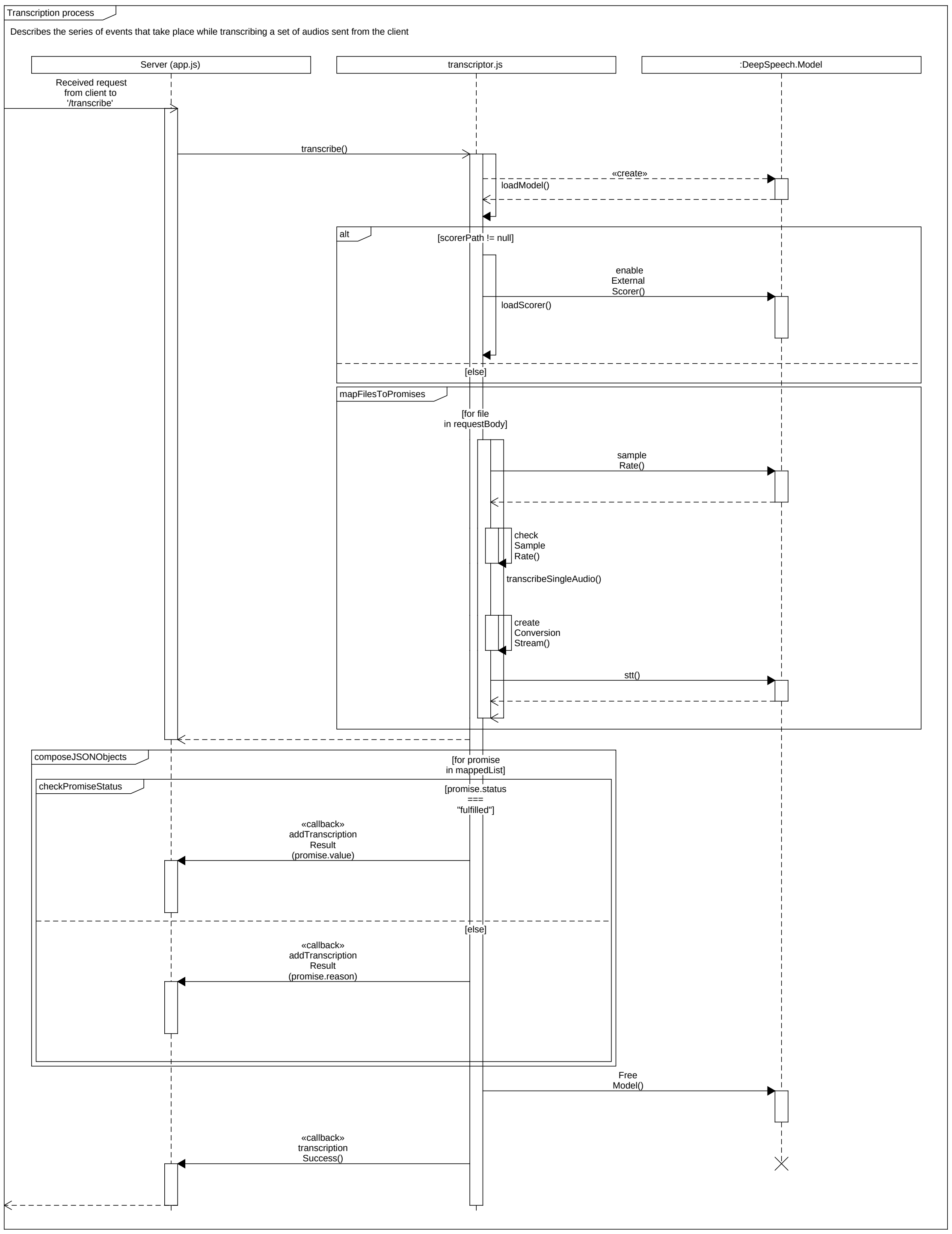

Transcription process
Describes the series of events that take place while transcribing a set of audios sent from the client
Server (app.js)
transcriptor.js
:DeepSpeech.Model
Received request from client to '/transcribe'
transcribe()
«create»
loadModel()
alt
[scorerPath != null]
enable External Scorer()
loadScorer()
[else]
mapFilesToPromises
[for file in requestBody]
sample Rate()
check Sample Rate()
transcribeSingleAudio()
create Conversion Stream()
stt()
composeJSONObjects
[for promise in mappedList]
checkPromiseStatus
[promise.status === "fulfilled"]
«callback» addTranscription Result (promise.value)
[else]
«callback» addTranscription Result (promise.reason)
Free Model()
«callback» transcription Success()

Figure 15: A UML Sequence Diagram showcasing the series of tasks performed by the backend service from receiving a request up to sending the response back

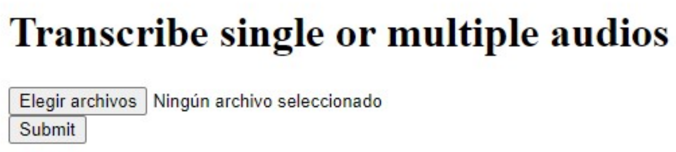
Figure 15, shows the overall function of the transcribe functionality. It explains the sequence of operations performed over the DeepSpeech Model object, as well as the intermediate functions that are executed in the transcriptor.js lifeline.

The series of files received by the client are put into separate threads, one for each file to be processed. This is done through the "Promise.allSettled" function. It creates a new flow of execution per received file and waits for the model to transcribe it. Then this promise is resolved or rejected (based on each case). However, the model only allows for sequential execution. Without having checked the implementation, it is known to use a lock-based approach (MUTEX) to prevent concurrent access to the model. This was discovered on accident by trying to implement the very same functionality I later discovered was already at play, so an extra implementation was not necessary at all.

After this execution, the list of Promises, whether they are rejected or fulfilled, are transformed into their respective output; value if fulfilled, reason of rejection otherwise. These get sent by a callback function to the app.js module that adds each of them to a list. Then, when every file is processed, another callback function is invoked to signal app.js to send the response with the list of configured JSON objects.

2.7.3 Use case scenarios

2.7.3.1 No files sent In figure 16, we can see a screenshot of the form used to send audios to the backend.


Transcribe single or multiple audios

Elegir archivos Ningún archivo seleccionado

Submit

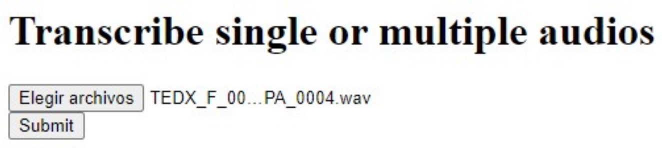
Figure 16: A screenshot of the form sent to the backend with no files attached

The backend returned the following response:

```
{"errorMessage":"No files provided"}
```

Listing 46: Expected message from the backend if no files are sent in the request

2.7.3.2 One file sent In figure 17, we can see a screenshot of the form used to send audios to the backend.


Transcribe single or multiple audios

Elegir archivos TEDX_F_00...PA_0004.wav

Submit

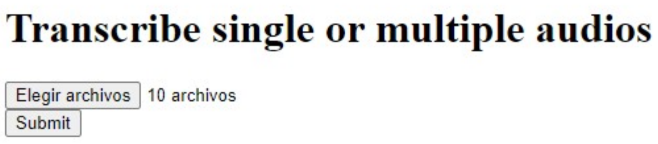
Figure 17: A screenshot of the form sent to the backend with one file attached

The backend returned the following response:

```
[{"audioFile":"TEDX_F_001_SPA_0004.wav","successful":true,"
   transcript":"sin empezar digo pues no le van mimarte las manos y
    los ojos no y despues","audioLength":6.8325000000000005}]
```

Listing 47: Expected shape of the response when sending a file to the backend. A list of JSON objects containing just one instance corresponding to the audio sent

2.7.3.3 Several files sent In figure 18, we can see a screenshot of the form used to send audios to the backend. In theory, there is no limit to the maximum amount of files that can be sent to the server. This could be set easily. One must note that the more files are sent to be transcribed at once, the longer the server takes to respond because these are processed sequentially.



Transcribe single or multiple audios

Figure 18: A screenshot of the form sent to the backend with several files attached

The backend returned the following response:

```
[{"audioFile":"TEDX_F_001_SPA_0001.wav","successful":true,"
    transcript":"y eso se para ím se se puede reducir en que en un
    despertar de la conciencia humana gracia","audioLength"
    :7.7598125},{"audioFile":"TEDX_F_001_SPA_0002.wav","successful":
    true,"transcript":"bueno les voy a cantar una de historia y ella
     es la vela","audioLength":6.0010625},{"audioFile":"
    TEDX_F_001_SPA_0003.wav","successful":true,"transcript":"si les
    ponen a una mujer al seenfrenta y le dicen el ñsealero escribir"
    ,"audioLength":7.695875},{"audioFile":"TEDX_F_001_SPA_0004.wav",
    "successful":true,"transcript":"sin empezar digo pues no le van
    mimarte las manos y los ojos no y despues","audioLength"
    :6.8325000000000005},{"audioFile":"TEDX_F_001_SPA_0005.wav","
    successful":true,"transcript":"yo no tengo nada en úcomn no
    nicotra e idioma este","audioLength":7.8238125},{"audioFile":"
    TEDX_F_001_SPA_0006.wav","successful":true,"transcript":"
    absolutamente nada es que somos mujeres somos seres humanos y
    entonces","audioLength":5.9264375000000005},{"audioFile":"
    TEDX_F_001_SPA_0007.wav","successful":true,"transcript":"me pone
     enfrente de ella en un mes entonces estaba trabajando en una
    organizacion internacional","audioLength":6.1396250000000006},{"
    audioFile":"TEDX_F_001_SPA_0008.wav","successful":true,"
    transcript":"e donde habiamos refugiados y ella es una de las
    refugiadas que estaba en el campo y es la caleta","audioLength"
    :8.164875},{"audioFile":"TEDX_F_001_SPA_0009.wav","successful":
    true,"transcript":"y entonces me pone en un grupo de ganas y yo
    como la maestra no","audioLength":5.8091875},{"audioFile":"
    TEDX_F_001_SPA_0010.wav","successful":true,"transcript":"cuando
    me doy cuenta y me enfrento que tus no sabia absolutamente nada
    y todo lo que habia aprendido","audioLength"
    :6.1290000000000004}]
```

Listing 48: Expected shape of the response when sending several files to the backend. A list of JSON objects containing n instances as many as audio files sent

3 Results

3.1 Metrics used to evaluate the results (WER and CER)

The CER and WER metrics are used to identify how closely a prediction resembles its target. These are easy to compute and give a summary of the quality of the recognition system [15].

3.1.1 Word Error Rate (WER)

Among others, the Word Error Rate metric, or WER for short, is used in speech recognition to measure the edit distance between the prediction and the target. It does that by considering the number of insertions, deletions, and substitutions, using the Levenshtein distance measure.

The WER metric is defined as follows:

$$WER = 100 * \frac{I + D + S}{N}$$

where

- I is the number of word insertions,
- D is the number of word deletions,
- S is the number of word substitutions, and
- N is the total number of words in the target.

3.1.2 Character Error Rate (CER)

When we manage character-based models for character-based languages, the error metrics tend to focus more on a character level. This is where we apply the Character Error Rate metric or CER for short. This is also referred to as Letter Error Rate or LER.

This CER metric is defined as follows:

$$CER = 100 * \frac{I + D + S}{N}$$

where

- I is the number of character insertions,
- D is the number of character deletions,
- S is the number of character substitutions, and
- N is the total number of characters in the target.

3.1.3 The issue with these metrics

WER and CER are useful to know how accurate our models are. The problem is that these do not provide any information on what the error was. Measuring specific types of errors would require conducting additional research to improve the models. An example of this would be Salient Word Error Rate, or SWER for short.

3.2 Training a Spanish model from scratch

The data used was taken from the West-Point Heroico Spanish dataset that did not make it into training. Plus, I added five samples of my creation, recording and introducing them into the dataset.

We performed a series of steps to create a testing batch for the final model:

- In a Windows machine, we created an anaconda Python virtual environment,
- We installed the DeepSpeech dependencies from the source code using the 'setup.py' script located in the root of the repository. This was performed using:

```
(deepspeech_test_phase) C:\Users\danal\git\DeepSpeech>python3 -
   m pip install .

```

Listing 49: Windows CMD command used to install DeepSpeech dependencies from the source into the Python venv

- We manually generated a small testing dataset with samples from the unused WPHSS_ES dataset and 5 samples created by me. We created the sample .csv file with the necessary format to be used by the engine. The file can be found under 'result-s/test_audios' named 'labels_sample.csv'. Here is a sample from that file:

```
wav_filename,wav_filesize,transcript
answers_99_92.wav,65512,no estoy sangrando
answers_99_91.wav,66952,no no tengo ninguna herida
[...]
usma_prompts_mexico_s9.wav,83220,ódnde áest el aeropuerto
usma_prompts_mexico_s10.wav,80024,ódnde áest la óestacin de
   tren
usma_prompts_nonnative_s198.wav,26170,ácul es su nombre
usma_prompts_nonnative_s199.wav,27554,ócmo se llama usted
[...]
custom_sample_01.wav,391280,esto es una prueba para el modelo
   de ótranscripcin
custom_sample_02.wav,315508,los resultados no tienen por équ
   ser buenos
custom_sample_03.wav,405620,étambin tiene problemas con acentos
    concretos del ñespaol

```

Listing 50: A sample from the testing dataset used to run the testing phase

- We ran the model from the existing checkpoints of the best model and instructed the engine to do only a test phase with our test sample dataset:

```
python3 DeepSpeech.py ^
    --n_hidden 2048 ^
    --alphabet_config_path "F:\OneDrive - Universidad de Oviedo
   \TFG\deepspeech_scorer\language_model\alphabet.txt" ^
    --test_files "F:/OneDrive - Universidad de Oviedo/TFG/
   results/test_audios/labels_sample.csv" ^
    --checkpoint_dir "F:/OneDrive - Universidad de Oviedo/TFG/
   ai_models/stt/checkpoints/" ^
    --scorer_path "F:/OneDrive - Universidad de Oviedo/TFG/
   deepspeech_scorer/esp_no_optimize.scorer" ^
    --test_batch_size 1 ^
    --test_output_file "F:/OneDrive - Universidad de Oviedo/TFG
   /results/test_phase_log.json" ^
```

```
    --report_count 10

```

Listing 51: Command used to run DeepSpeech only to do a testing phase over the existing model using the small data sample provided

Note that we must pass some settings as parameters to make this work: The same network geometry with '–n_hidden' and the alphabet the model was trained with so the output layer has the same dimensions as that saved in the checkpoints ('–alphabet_config_path'),

- The engine generates a log file with data from the results obtained in the testing phase. This file can be found under 'results' named 'test_phase_log.json'. Here is a sample from that file:

```
[
    {
        "wav_filename": "F:\\OneDrive - Universidad de Oviedo\\
    TFG\\results\\test_audios\\usma_prompts_nonnative_s203.wav
    ",
        "src": "busca alrededor de la mesa",
        "res": "busca alrededor de la mesa",
        "loss": 24.437515258789062,
        "char_distance": 0,
        "char_length": 26,
        "word_distance": 0,
        "word_length": 5,
        "cer": 0.0,
        "wer": 0.0
    },
    {
        "wav_filename": "F:\\OneDrive - Universidad de Oviedo\\
    TFG\\results\\test_audios\\usma_prompts_mexico_s3.wav",
        "src": "eres de los estados unidos verdad",
        "res": "eres de los estados unidos verdad",
        "loss": 8.334741592407227,
        "char_distance": 0,
        "char_length": 33,
        "word_distance": 0,
        "word_length": 6,
        "cer": 0.0,
        "wer": 0.0
    },
    [...]

```

Listing 52: A sample from the .json log file created by the training engine after finishing the test phase to obtain results

To perform this test phase using the trained model's checkpoints, we had to modify the checkpoint files to fix the file paths:

```
-> F:\OneDrive - Universidad de Oviedo\TFG\ai_models\stt\
    checkpoints\best_dev_checkpoint
FROM:
model_checkpoint_path: "/checkpoints/best_dev-329670"
all_model_checkpoint_paths: "/checkpoints/best_dev-329670"

TO:
```

```
model_checkpoint_path: "./best_dev-329670"
all_model_checkpoint_paths: "./best_dev-329670"

-> F:\OneDrive - Universidad de Oviedo\TFG\ai_models\stt\
    checkpoints\checkpoint
FROM:
model_checkpoint_path: "/checkpoints/train-344775"
all_model_checkpoint_paths: "/checkpoints/train-339851"
all_model_checkpoint_paths: "/checkpoints/train-340509"
all_model_checkpoint_paths: "/checkpoints/train-340659"
all_model_checkpoint_paths: "/checkpoints/train-343129"
all_model_checkpoint_paths: "/checkpoints/train-344775"

TO:
model_checkpoint_path: "./train-344775"
all_model_checkpoint_paths: "./train-339851"
all_model_checkpoint_paths: "./train-340509"
all_model_checkpoint_paths: "./train-340659"
all_model_checkpoint_paths: "./train-343129"
all_model_checkpoint_paths: "./train-344775"
```

Listing 53: STT Training checkpoint index files modifications to work with the testing phase in Windows host

3.2.1 Results obtained without a scorer file

These results are obtained from the test phase of the model not using a scorer file. That is, the model is transcribing the test audio files using only the acoustic model. Lacking some capacity to improve the results based on the language model, not accessible during this testing phase. Look at table 1 to see the results.

3.2.2 Results obtained with an English scorer file

To stress the relevancy of using a scorer made from a corpus from the target language, we have run the same testing phase, but this time we will use the English scorer file provided by the DeepSpeech engine in their releases page for tag 0.8.0: deepspeech-0.8.0-models.scorer. Look at table 2 to see the results.

3.2.3 Results obtained with our Spanish scorer file

These results are obtained using the created scorer file, which improves the model's ability to transcribe and adjust the output of the acoustic model. Look at table 3 to see the results.

WAV Filename	Source transcript	Result transcript	WER	CER
usma_prompts_mexico_s7.wav	buena suerte	buena suerte	0.0	0.0
usma_prompts_mexico_s1.wav	vivo en una casa	vivo en una casa	0.0	0.0
answers_96_27.wav	el tono de su voz es muy grave	el tono de su boz es muy grave	0.125	0.033
usma_prompts_mexico_s3.wav	eres de los estados unidos verdad	eres de los estados unidos noerdad	0.167	0.061
answers_98_61.wav	aproximadamente unas tres horas y media	de aproximadamente unas tres horas y media	0.167	0.077
answers_99_91.wav	no no tengo ninguna herida	o no tengo ninguna herida	0.2	0.038
usma_prompts_mexico_s6.wav	qué le vaya bien	que le vaya bien	0.25	0.062
answers_98_58.wav	claro me esperan mis familiares ahí es donde voy a hospedarme	claro me esperan mis amiliares ahó es donde voy a ospedarme	0.273	0.049
usma_prompts_mexico_s9.wav	dónde está el aeropuerto	tonde esté el aeropuerto	0.5	0.125
usma_prompts_nonnative_s205.wav	me da lo mismo	pecra lo mismo	0.5	0.214
usma_prompts_mexico_s10.wav	dónde está la estación de tren	tonde esté a istaciún de tren	0.667	0.2
usma_prompts_nonnative_s201.wav	viven aquí desde los años setenta	i gre quite es los años setenta	0.667	0.364
usma_prompts_mexico_s4.wav	soy de méxico	soy demíxico	0.667	0.154
usma_prompts_nonnative_s198.wav	cuál es su nombre	cual es un obre	0.75	0.294
usma_prompts_nonnative_s203.wav	busca alrededor de la mesa	posca a redededor de la pesa	0.8	0.231
usma_prompts_nonnative_s200.wav	cuál es su numero de teléfono	cuando su numoro dicelafoo	0.833	0.379
usma_prompts_nonnative_s206.wav	a qué hora es el examen	aqullahor es la elica	0.833	0.522
custom_sample_04.wav	cómo se comportará con alguna pregunta	omosecon porana con ra pordado pero	1.0	0.605
custom_sample_05.wav	sin embargo los resultados son más que aceptables	enra bar o lstatos surmrmas quea eltals	1.0	0.51
custom_sample_02.wav	los resultados no tienen por qué ser buenos	grs fortadas e nor enean por ese recoal	1.0	0.558
usma_prompts_mexico_s8.wav	dónde está el banco	donde estoy nanco	1.0	0.316
usma_prompts_nonnative_s199.wav	cómo se llama usted	como soylamos	1.0	0.526
usma_prompts_mexico_s2.wav	dónde vives tu	todnde vive est	1.0	0.5
answers_96_70.wav	mis pasajeros extraviaron sus carnet de identidad	mi s pasajeros estraavieron su carneter intidad	1.0	0.204
usma_prompts_nonnative_s202.wav	hazme reír	hasta re	1.0	0.6
usma_prompts_mexico_s5.wav	de dónde es usted	ela nesusted	1.0	0.471
usma_prompts_nonnative_s204.wav	enrique canta mal	querrique y cantama	1.0	0.412
answers_99_92.wav	no estoy sangrando	no esto es angrando	1.0	0.167
custom_sample_01.wav	esto es una prueba para el modelo de transcripción	es tradaves ona propea ga para lar el mordrel nao de teral que sieran	1.222	0.64
custom_sample_03.wav	también tiene problemas con acentos concretos del español	tarmpiern ueendertura orle mas corrads e ntdos conn grertos de espaenia	1.375	0.526

Table 1: Contains the results from running a test phase using no scorer file over the small testing dataset

WAV Filename	**Source transcript**	**Result transcript**	**WER**	**CER**
usma_prompts_mexico_s7.wav	buena suerte	buena suerte	0.0	0.0
usma_prompts_mexico_s1.wav	vivo en una casa	vivo en una casa	0.0	0.0
answers_96_27.wav	el tono de su voz es muy grave	el toro de su voz es muy grave	0.125	0.033
usma_prompts_mexico_s3.wav	eres de los estados unidos verdad	eres de los estados unidos nordad	0.167	0.061
answers_99_91.wav	no no tengo ninguna herida	no tengo ninguna herida	0.2	0.115
answers_98_58.wav	claro me esperan mis familiares ahí es donde voy a hospedarme	claro me espera mis familiares ae es donde voy a speare	0.273	0.115
usma_prompts_mexico_s4.wav	soy de méxico	soy de mexico	0.333	0.077
usma_prompts_nonnative_s199.wav	cómo se llama usted	como se llama	0.5	0.316
usma_prompts_mexico_s6.wav	qué le vaya bien	que le va bien	0.5	0.188
usma_prompts_nonnative_s203.wav	busca alrededor de la mesa	scared trode la mesa	0.6	0.346
usma_prompts_mexico_s10.wav	dónde está la estación de tren	donde esta estacion de tren	0.667	0.167
usma_prompts_nonnative_s204.wav	enrique canta mal	queried canta	0.667	0.588
answers_98_61.wav	aproximadamente unas tres horas y media	a proxima mente unas tres horas media	0.667	0.128
answers_96_70.wav	mis pasajeros extraviaron sus carnet de identidad	mis pasagero savaron su carne terentia	0.857	0.306
custom_sample_01.wav	esto es una prueba para el modelo de transcripción	es trades on proper are more nateral question	0.889	0.58
custom_sample_04.wav	cómo se comportará con alguna pregunta	a nasconora concord	1.0	0.711
custom_sample_05.wav	sin embargo los resultados son más que aceptables	erostratos masquerades	1.0	0.653
custom_sample_03.wav	también tiene problemas con acentos concretos del español	amintor lamas orlandos congresses	1.0	0.561
custom_sample_02.wav	los resultados no tienen por qué ser buenos	as for cards or near sea	1.0	0.721
usma_prompts_nonnative_s200.wav	cuál es su numero de teléfono	no sun nor disease	1.0	0.724
usma_prompts_mexico_s8.wav	dónde está el banco	donde esta manco	1.0	0.316
usma_prompts_mexico_s9.wav	dónde está el aeropuerto	donde esta el aero puerto	1.0	0.125
usma_prompts_nonnative_s198.wav	cuál es su nombre	jules nor	1.0	0.529
usma_prompts_nonnative_s201.wav	viven aquí desde los años setenta	i requite gosos tent	1.0	0.576
usma_prompts_nonnative_s206.wav	a qué hora es el examen	clare a la	1.0	0.783
usma_prompts_mexico_s2.wav	dónde vives tu	donde divest	1.0	0.286
usma_prompts_nonnative_s202.wav	hazme reír	are	1.0	0.7
usma_prompts_mexico_s5.wav	de dónde es usted	e a nested	1.0	0.529
answers_99_92.wav	no estoy sangrando	no esto es agrando	1.0	0.222
usma_prompts_nonnative_s205.wav	me da lo mismo	taoism	1.0	0.643

Table 2: Contains the results from running a test phase using the DeepSpeech releases scorer file over the small testing dataset

WAV Filename	Source transcript	Result transcript	WER	CER
usma_prompts_nonnative_s203.wav	busca alrededor de la mesa	busca alrededor de la mesa	0.0	0.0
usma_prompts_mexico_s3.wav	eres de los estados unidos verdad	eres de los estados unidos verdad	0.0	0.0
answers_96_27.wav	el tono de su voz es muy grave	el tono de su voz es muy grave	0.0	0.0
usma_prompts_mexico_s7.wav	buena suerte	buena suerte	0.0	0.0
usma_prompts_mexico_s1.wav	vivo en una casa	vivo en una casa	0.0	0.0
answers_98_58.wav	claro me esperan mis familiares ahí es donde voy a hospedarme	claro me esperan mis familiares ahi es donde voy a hospedarme	0.091	0.016
answers_98_61.wav	aproximadamente unas tres horas y media	de aproximadamente unas tres horas y media	0.167	0.077
answers_99_91.wav	no no tengo ninguna herida	no tengo ninguna herida	0.2	0.115
usma_prompts_mexico_s6.wav	qué le vaya bien	que le vaya bien	0.25	0.062
usma_prompts_mexico_s4.wav	soy de méxico	soy de mexico	0.333	0.077
answers_96_70.wav	mis pasajeros extraviaron sus carnet de identidad	mis pasajeros estuvieron su carnet de entidad	0.429	0.143
usma_prompts_mexico_s10.wav	dónde está la estación de tren	donde esta la estacion de tren	0.5	0.1
usma_prompts_mexico_s9.wav	dónde está el aeropuerto	donde esta el aeropuerto	0.5	0.083
usma_prompts_nonnative_s199.wav	cómo se llama usted	como se llama	0.5	0.316
usma_prompts_nonnative_s201.wav	viven aquí desde los años setenta	si requieres los años setenta	0.5	0.333
custom_sample_01.wav	esto es una prueba para el modelo de transcripción	es es una propiedad para el modelo de el que era	0.556	0.4
usma_prompts_nonnative_s200.wav	cuál es su numero de teléfono	cuando su numero dice las	0.667	0.483
usma_prompts_mexico_s2.wav	dónde vives tu	donde vives	0.667	0.286
answers_99_92.wav	no estoy sangrando	no esto es sangrando	0.667	0.167
usma_prompts_mexico_s8.wav	dónde está el banco	donde este banco	0.75	0.211
usma_prompts_nonnative_s198.wav	cuál es su nombre	cual es una obra	0.75	0.412
usma_prompts_mexico_s5.wav	de dónde es usted	era un usted	0.75	0.529
custom_sample_04.wav	cómo se comportará con alguna pregunta	mascarada con acordado pero	0.833	0.658
custom_sample_03.wav	también tiene problemas con acentos concretos del español	tarentola lemas corramos con restos de esta	0.875	0.456
custom_sample_05.wav	sin embargo los resultados son más que aceptables	en otras subastas	1.0	0.735
custom_sample_02.wav	los resultados no tienen por qué ser buenos	portadas de empresas	1.0	0.698
usma_prompts_nonnative_s206.wav	a qué hora es el examen	aclare la alicia	1.0	0.739
usma_prompts_nonnative_s202.wav	hazme reír	hasta	1.0	0.7
usma_prompts_nonnative_s204.wav	enrique canta mal	queria cantaba	1.0	0.529
usma_prompts_nonnative_s205.wav	me da lo mismo	mecanismo	1.0	0.5

Table 3: Contains the results from running a test phase using our Spanish scorer file over the small testing dataset

Now, here is a plot of the results obtained in the three testing sessions shown using boxplots:

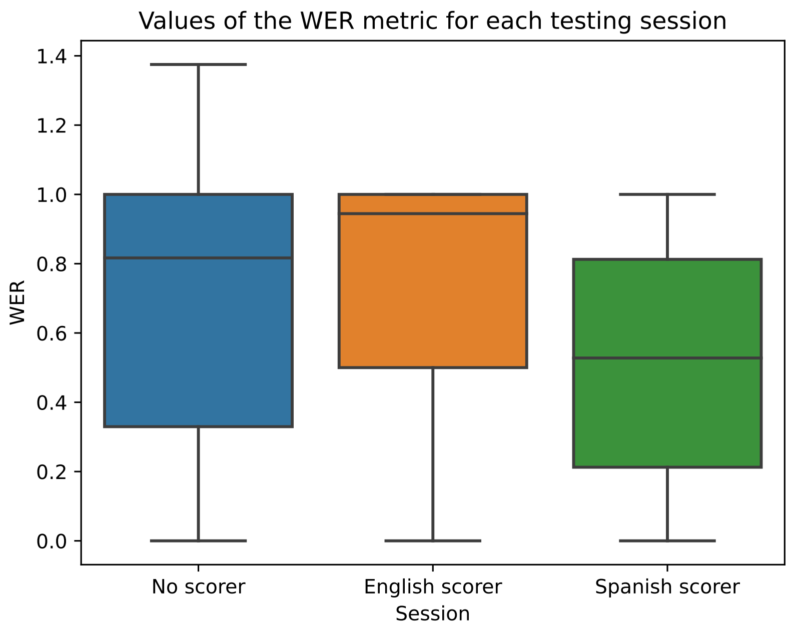


Figure 19: A boxplot showing the values obtained for the WER metric used to test the model against a small dataset containing 30 samples

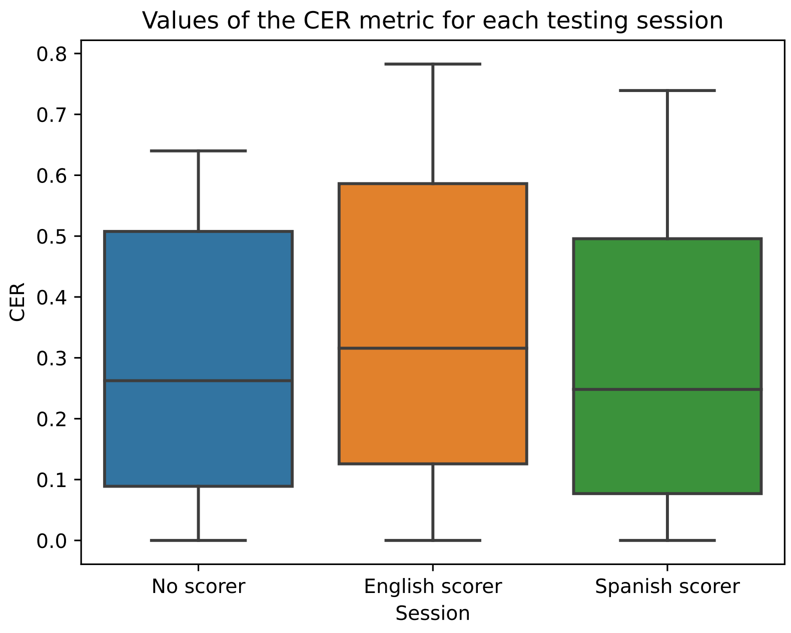


Figure 20: A boxplot showing the values obtained for the CER metric used to test the model against a small dataset containing 30 samples

The values shown in figure 19 correspond to the WER metric from the testing phase for each testing session. The same is included for the CER metric in figure 20.

4 Discussion of the results

The whole process described in this work has many steps. Each of them has suffered from a lot of things related to the learning process. Overall knowledge of the project was a major issue during this undertaking. The results are enough to demonstrate the potential of this framework, given good-quality data and meticulous cleaning steps.

The results are kind of mediocre with new speech. This should not come as a surprise to the reader as this document resembles the first contact with any machine learning process or framework ever. However, the process has created a deeper understanding of the engine, and it would be easier to perform the training process again, should the resources be available or newer iterations needed.

As shown in the previous section (see section 3.2), the results obtained from the three testing phases we performed showed the performance of the model under several circumstances. All the data used for testing was new, unseen data by the model.

4.1 Evaluating our model results

Table 4 shows the average values of WER and CER metrics obtained for each test performed.

	Avg. WER	**Avg. CER**
No scorer	0.700	0.295
English scorer	0.715	0.370
Spanish scorer	0.533	0.294

Table 4: Contains the WER and CER average values obtained from the different testing sessions performed

Although it can be easily understood from the data, it is of utmost importance to use a well-trained scorer for the target language. The acoustic model on its own might not be enough to create good-quality transcriptions.

Now that we have obtained a set of results from the model, we can assess its quality by following the criteria from Microsoft [9].

- A WER value of up to 10% provides evidence that the model has good-quality and that it is ready to be implemented into the target system or application,
- A WER value ranging between 10% and 20% is acceptable, but this suggests that some extra training might be needed,
- A WER value higher than 20% yields a faulty model with poor quality. This suggests that we need more data or that we need to train the model for longer periods of time.

Following the guidelines of the mentioned page, and looking at results of our model in table 4, we have to say that our model is not fit for general use in the target domain due to being of poor quality with an average WER value of 53%.

However, this is biased by the number of samples and audio files used in the testing phase.

4.2 Ways to improve this results

The results can be improved using the following means [5], we can list the components needed to obtain models with better quality:

- **Increase the size of the dataset**: This kind of software is based around supervised and autonomous learning. Meaning the model will use input training data to learn how to perform the task at hand (STT in this case). Simply put, the more samples it has to work with, the better it gets when making predictions. With lots of data comes the necessity for computing power to process all the data in a rather short amount of time,
- **Improve data balance**: Meaning we need to represent our universe of discourse in the training datasets. If we aim to transcribe audios in noisy environments, we should train with noisy data to show the model how to generalize those samples in real transcription scenarios. For a general-purpose transcriptor, we need to input both women and men training samples, various language accents and several voice pitches and timbres,
- **Increase the variety of environmental conditions of the dataset**: The model needs to be trained and tested around various and controlled acoustic conditions to ensure that it has a stable performance with new input data obtained from different recording sources.

5 Future work

Now that the project has been almost completely explained, here are some ideas that were devised to extend the functionality of the project. These ideas were meant to be implemented on-site, something that was not possible due to time and resource-bound constraints.

Bear in mind that, in most sections, there are diagrams included to gain insight into the solutions proposed, and to support the explanations provided in the text.

- Firstly, a Machine Learning training technique called Transfer Learning is applied to the project. This section explains how to achieve this and how to configure the DeepSpeech engine to perform this alternate training mode.
- Secondly, we explain the approach we would follow to generate more training data, should the opportunity had been available. With a diagram, we explain the different cases we can operate on and the techniques used to approach this solution.
- In addition, a third section featuring an extension to the STT backend is proposed using WebSockets. This extension would be crucial to projects that need a real-time transcription.
- Last but not least, in the Online Learning section, we will present a small and rather simple architecture of microservices to train new models from gathered data automatically.

5.1 Transfer learning from English to Spanish

This technique called Transfer Learning is an alternative to training a model from scratch since it allows the user to continue training a model based on the training process of an existing one just adjusting the parameters related to (usually) the output layer. The rest of the architecture of the NN would remain the same, as well as the existing fine-tuned parameters (weights, biases, ...). The rest of the parameters would be corrected during the transfer-learning process thanks to backpropagation. This is also a convenient alternative if we want to train a model for a language with a different alphabet than the one used as a transfer-learning base. In this case, we present a change from English to Spanish. These two have different alphabets, so the output layer does not match our target language.

Should it be needed, the engine accepts the drop of n layers in the range of 1 to 5. The most sensible value should be one, so just the output layer is dropped, but it is configurable through the '–drop_source_layers [VALUE]' flag.

For this process to take place, we need the following:

- The set of checkpoints from the release model we want to resume training from,
  In this case, these checkpoints can be downloaded from the releases page of the DeepSpeech repository on GitHub. Simply browse the specific version of the engine we want to use and download the following file: deepspeech-0.8.0-checkpoint.tar.gz.
- A place to store the new checkpoints of the final model,
- A .txt file containing the new alphabet, one character per line as in the following example (alphabet.txt file located under /DeepSpeech/data/alphabet.txt)

```
# Each line in this file represents the Unicode codepoint (UTF
   -8 encoded)
# associated with a numeric label.
# A line that starts with # is a comment. You can escape it
   with \# if you wish
```

```
# to use '#' as a label.

a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x
y
z
'
# The last (non-comment) line needs to end with a new line.

```

Listing 54: alphabet.txt this file contains the alphabet with which the model is going to be trained

Once we have a hold of these resources, we can prepare for training the model. Following the project's documentation [21], we can train a new model following this approach using this command:

```
~/DeepSpeech$ python DeepSpeech.py \
    --drop_source_layers 1 \
    --alphabet_config_path my-new-language-alphabet.txt \
    --save_checkpoint_dir path/to/output-checkpoint/folder \
    --load_checkpoint_dir path/to/release-checkpoint/folder \
    --train_files my-new-language-train.csv \
    --dev_files my-new-language-dev.csv \
    --test_files my-new-language-test.csv
```

Listing 55: Bash command used to execute the training engine following a Transfer learning approach also loading resources from the checkpoints directory to load the previous training progress

We can list several benefits from following this approach:

- Training time is significantly reduced,
- Not that much data needed as if we trained the model from scratch,
- If the checkpoints used as a base are from training a model with decent accuracy, our target model is likely to be accurate if we use high-quality and sanitized data,

- Also, resource constraints should not be a problem since the engine is built and has been tested to work in ranges from high computing power servers to just a Raspberry Pi 4.

5.2 Data augmentation on the training data

Data augmentation is often used when there is not enough data to train a model. The basis is to combine, modify or adapt the existing data to generate more samples to train the model. Depending on the universe of discourse this can be challenging. In this case, we are managing audio files. The representations are sound-wave-based.

5.2.1 Custom-made approach

As this is not something implemented into this work, this section is more of an untested hypothesis than an actual proof-of-concept. At first, this approach is simple enough to be feasible, but there can always be technicalities that make it hard. So here are a couple of ideas to generate new samples to engorge the training datasets:

1. We would generate pairs of audio files by checking their duration. These audios should be close to the same duration, or one of them is longer than the other.
2. Having these pairs set up, we lower the volume of one of the audios to around 50% or less so that it behaves as noise on top of the "actual" target audio. The idea is to lower the volume of the shorter one (in case these are not the same duration).
3. We now label the new pair with the original label from the longer audio (or the one with the original volume, if they are not the same duration).

Now, this can work both ways if the audios are close to the same duration. So we can double the amount of data by performing these steps over the two audios in each pair. In case there is a pair of audios that is unmatched in duration, we can simply cut the longer audio to match the target audio. In this case, losing information on the cut audio does not matter because it behaves as noise, and therefore is not bound to any target label.

A very similar approach could be to use several audios with very low volume acting as noise on top of another audio, this being the target to transcribe. This way, using file combinations, we can create a massive dataset just with a couple of thousands of audio files.

The summary of this idea is to generate combinations of audios, playing with noise-acting and target-acting, volumes of the tracks, and manage the labels correctly. This would help the model build resistance towards noisy environments. Which, in most cases, is the needed result. It is hard to use a model that requires a silent environment or good audio quality. See figure 21 for a more visual explanation.

This image shows the four cases that could apply to the pairs of audios created from the datasets. The ranges of volumes set for each audio are merely arbitrary. With some testing, this value could be fine-tuned to approximate the audio to be just noise and not confuse the model. To explain the image:

- The coloured rectangles in the image represent each audio of the pair.
  - A red background colour means the audio is meant to act as noise.
  - A green background colour means the audio is meant to act as the actual training audio with its true label attached.

- The black-bordered rectangles represent each of the cases we can find within the dataset.
- The duration of each audio is represented as the width of its respective box. Again, this is completely arbitrary and it is used as a means to communicate the approach. In reality, there could be the case where no two audios have the same length or that all of them are the same length.

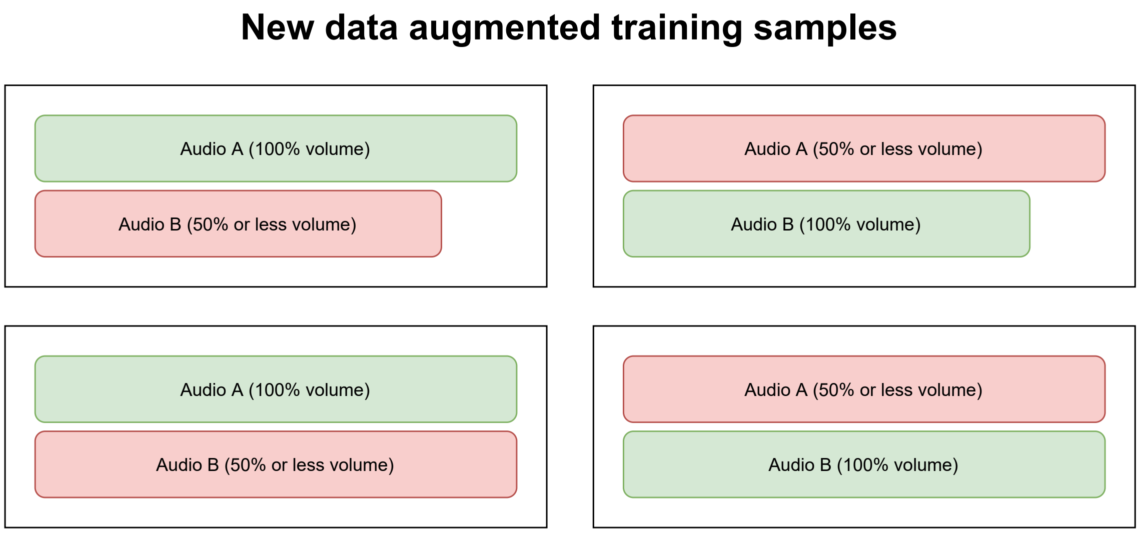

New data augmented training samples
Audio A (100% volume)
Audio B (50% or less volume)
Audio A (50% or less volume)
Audio B (100% volume)
Audio A (100% volume)
Audio B (50% or less volume)
Audio A (50% or less volume)
Audio B (100% volume)

Figure 21: A simple diagram showing the four possible cases of data augmentation depending on the duration of the two audios in each pair.

5.2.2 DeepSpeech engine augment approach

The DeepSpeech command line also has a tool to generate in-place data augmentation following several augmentation types.

These augments are added to the training pipeline by using the '–augment' flag as follows:

```
[...]
--augment augmentation_type1[param1=value1,param2=value2,...]
--augment augmentation_type2[param1=value1,param2=value2,...]
[...]
```

Listing 56: Bash flag used to add data augment phases to the training data using the DeepSpeech.py script

For instance, using the example in the documentation, one can add an overlay to the training audio samples by adding this flag to the training command:

```
--augment overlay[p=0.1,source=/path/to/audio.sdb,snr=20.0] ...
```

Listing 57: Bash flag used to add an overlay augment to the input data of the DeepSpeech.py script

However, this type of data augmentation is complicated and would require separate research to analyze which are the best augments to apply to the input data.

There are several upgrades we can make, like pre-creating the augmented set and then training with that new set. Plus, the values in the '–augment' flag's syntax can take different formats to specify ranges, simple values, and a combination of both.

5.3 Extension to the backend prototype using WebSockets

This model we have created can be used through the command line with the deep-speech utility or a package API. This last option was used to implement the backend explained in section 2.7.

Taking advantage of these APIs available for many programming languages, and reading the documentation, we can see that the Model object works with audio streams. Now, what if we wanted to do a real-time transcription through this same backend?

In figure 22, we can see a sequence diagram showcasing how the Client would interact with our server through the WebSocket technology. As these allow for binary data transport, we can use them to send the recorded buffers to the server so they can be processed by our existing transcription module.

The WebSockets are an event-based tool that allows for the bidirectional communication between two parties that exchange data through the network using the "ws://" (not cyphered) or "wss://" (cyphered) protocols. It is widely used by many web applications all across the internet and is supported by most web browsers.

Some execution flows in the diagram are executed when specific events are fired. These arrows should be supposed to have an event attached (we assume an event fired them). Moreover, to simplify the diagram, any error handling has been omitted, but these should be in place if the feature described was to be implemented.

Client-side operations would follow this process:

1. User requests the page from the server that holds the real-time transcription feature,
2. User asks the page to start a new STT session (i.e. clicking a button),
3. The page shows some indication that the recording of live audio has started and shows an empty text area to write the transcription on,
4. Behind the scenes, we request the server for an upgraded connection to use WebSockets,
5. Now, with an established connection, the recording audio is being written onto a buffer,
6. Each second of recorded audio (arbitrary quantity), the buffer is written onto the WebSocket object created during the handshake for the server to process,
7. Once the server is finished transcribing the buffer, a new message should be sent into the WebSocket for us to process,
8. We then append the message's contents (transcript) to the text area showing on the screen for the user to see,
9. Repeat the process from the 5th step until the user finishes the STT session (i.e. clicking a button),
10. The process is finished and the socket object is closed and released,
11. The user may now start a new session by repeating this process from step 2.

Server-side operations would follow this process:

1. The server awaits requests from the client,
2. Receives a request for the real-time transcription page and sends the necessary resources for rendering it in the client,
3. The server now receives a request for an upgraded HTTP connection to use WebSockets,
4. Connection established and channel created,
5. The socket fires an event indicating a new message has been received,

6. The server sends the buffer contained in the message to the transcription module,
7. The transcription module then saves the content into a file and transcribes it using the transcribe() function explained in this section: 2.7.2 with the sequence diagram shown in figure 15,
8. Once the transcriptor module is finished, the data is sent back through a callback function to the server,
9. The server then writes the contents of the transcription into the socket for the client to process,
10. Repeat this process for every new message from the 5th step.

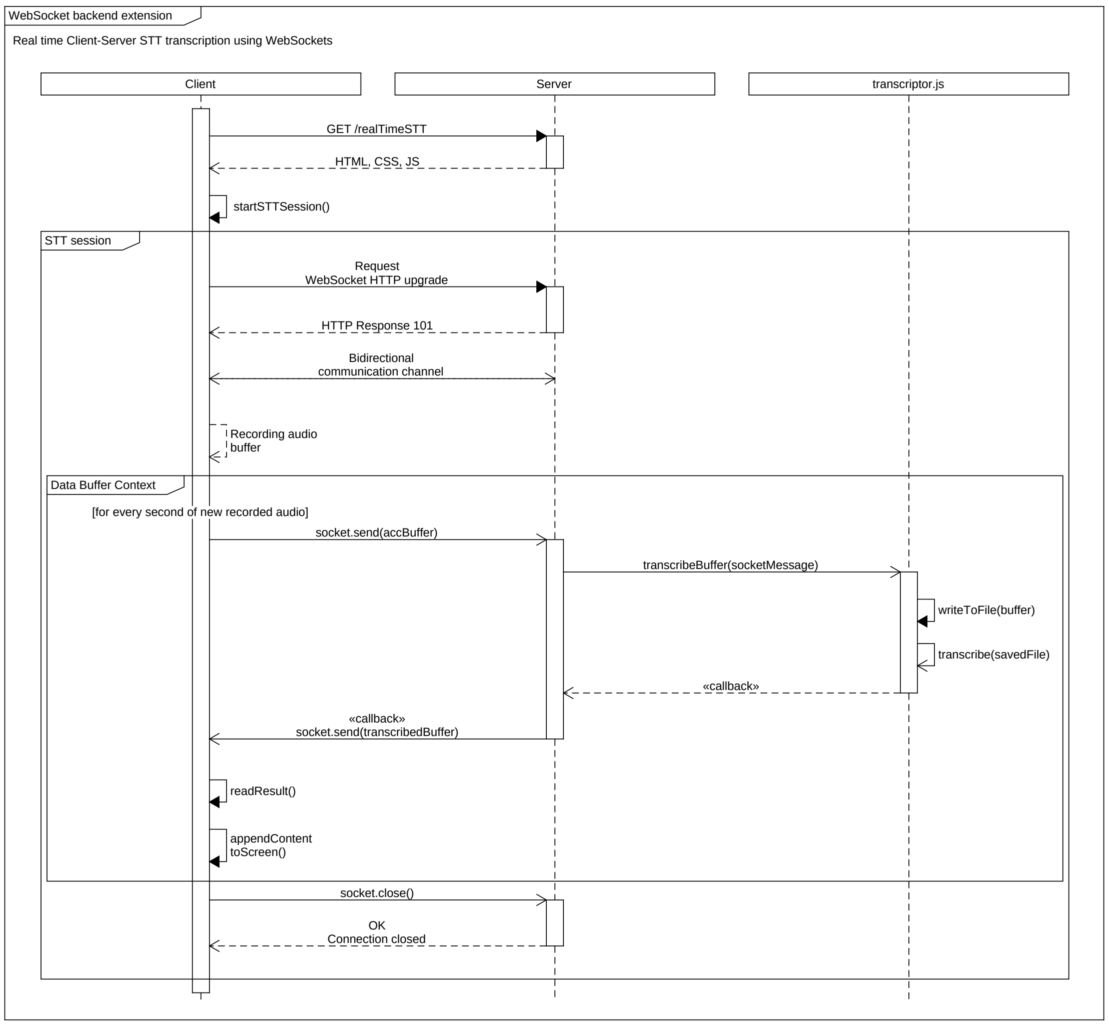


Figure 22: A UML Sequence Diagram showing how Clients would interact with the server when using the real-time transcription feature

5.4 Integration into a CI/CD pipeline for online learning

Online Training is a technique used in Machine Learning to fine-tune already trained models using new data. This technique is limited to some ML techniques. To the scope of this project, Neural Networks can be Online Trained given some conditions.

This approach is possible thanks to the implementation of the DeepSpeech engine. The main training program saves training checkpoints during the process. This allows us to continue training from these checkpoint files should anything happen to the system, runtime errors or decide to manually stop early, for instance. In our case, we simply want to resume the training process without losing the time invested to get to that point.

We assume that the new data that comes into the Platform is provided by pairs of audio utterances and labels attached to each of them. These audios represent the new training content.

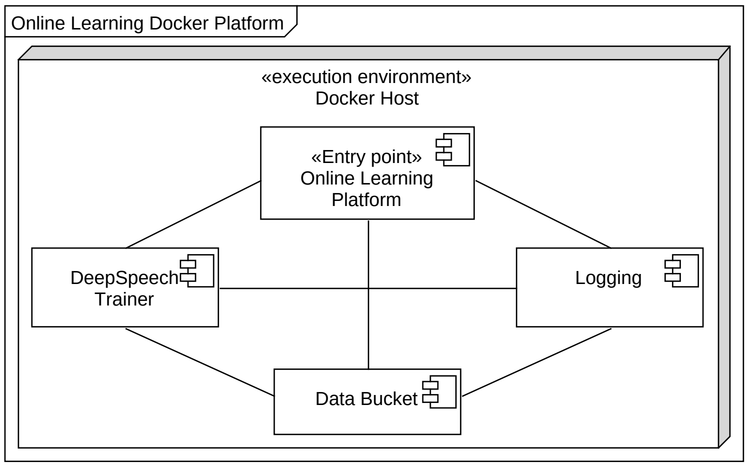

Online Learning Docker Platform
«execution environment» Docker Host
«Entry point» Online Learning Platform
DeepSpeech Trainer
Logging
Data Bucket

Figure 23: A UML deployment diagram to show the different components that would compose the proposed Online Learning Platform

This proposal is nothing but a complement to what has been stated previously in this document. Taking advantage of technologies such as Docker, Kubernetes and Docker Swarm, we can implement a system that could be integrated into a CI/CD pipeline and perform Online Training to continuously deploy new models with each session. For reference, see diagram 23 to see the structure of the Online Training Platform (OTP). We would describe this pipeline as follows:

1. We assume there is a service gathering audio files and labelling them using a third-party service or software such as Google Cloud's,
2. This service should be sending audio batches periodically to the OTP,
3. The OTP then answers by triggering the sequence of tasks described in the diagram shown in figure 24. To serve as a summary (assuming logging is in place, hence omitting that part):
   a) Receive new files from external service,
   b) Store these files in the data bucket component,
   c) Signal training module new files have been received.
   d) Check if we can start a new training session. If so, continue, else, jump to the last step
   e) Run data preprocessing on the input data,
   f) Run the training engine from the previous checkpoint,
   g) When the engine has finished training: store output model, checkpoints, log files,

   h) Notify the OTP that the training session has finished,
   i) Finished interaction with the OTP.

4. If the training component trained a new model and finished successfully, the OTP component will be notified. This message will then be relayed to any observer informing that a new model release is available, optionally deploying it in any STT environment, should it be needed.

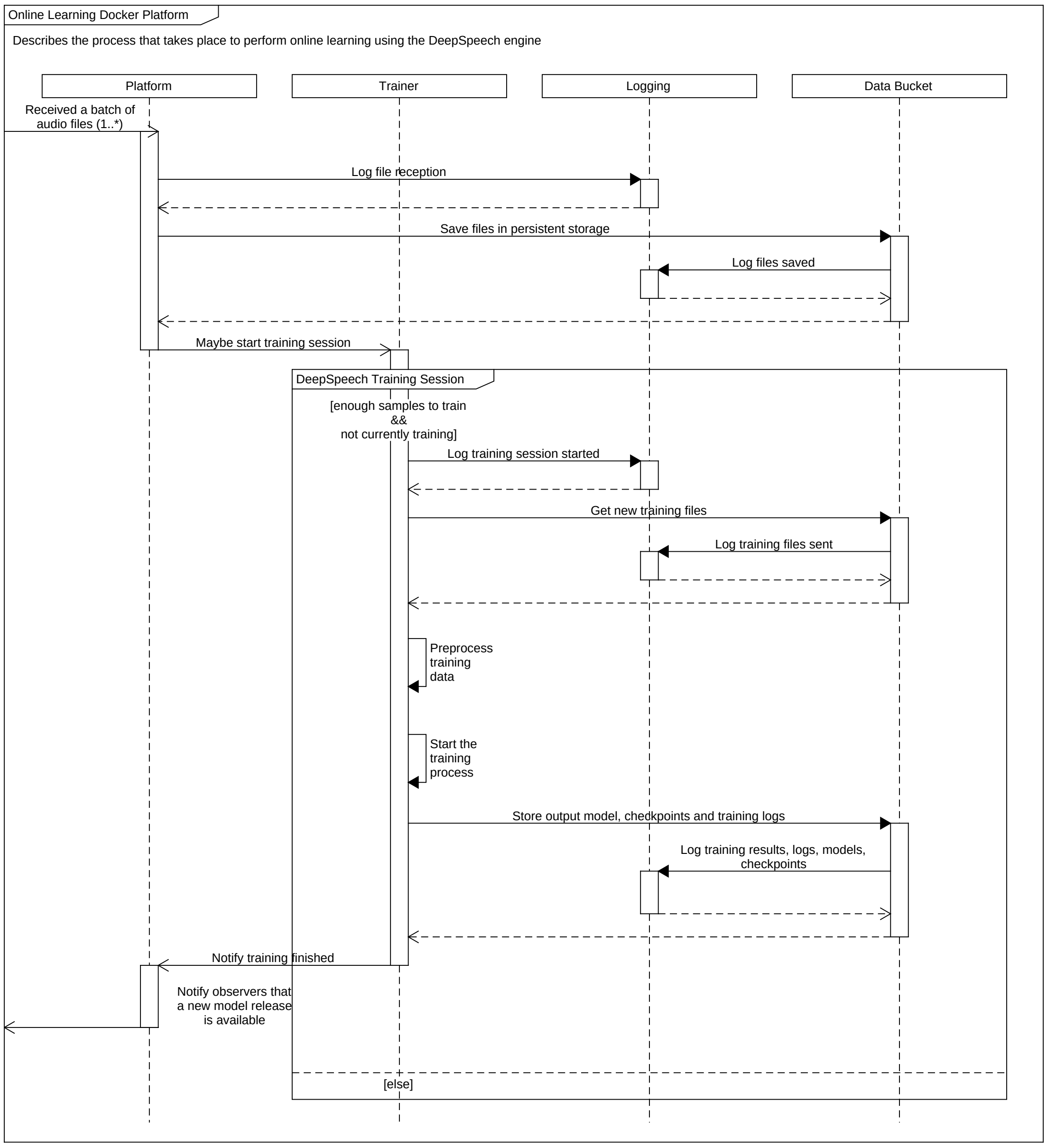

Online Learning Docker Platform
Describes the process that takes place to perform online learning using the DeepSpeech engine
Platform
Trainer
Logging
Data Bucket
Received a batch of audio files (1..*)
Log file reception
Save files in persistent storage
Log files saved
Maybe start training session
DeepSpeech Training Session
[enough samples to train && not currently training]
Log training session started
Get new training files
Log training files sent
Preprocess training data
Start the training process
Store output model, checkpoints and training logs
Log training results, logs, models, checkpoints
Notify training finished
Notify observers that a new model release is available
[else]

Figure 24: A UML Sequence diagram showing the planned process to follow when new files are sent to the Online Training Platform

6 Conclusions

This project arose from the need to replace cloud-based transcription services with a proprietary solution. The outcomes of this project have been quite insightful as they have provided great amounts of knowledge in the NLP and STT Machine Learning worlds.

In addition, among the contributions made by this undertaking to my personal experience, I can point out:

- I obtained new and exciting knowledge on Artificial Intelligence procedures, which brought a more open-minded setup to facing the problems that arose during this project and those to come,
- Also, I have improved my ability to walk through pages of documentation, forums, and comments on public repositories of an open-source project. Moreover, the Mozilla Discourse forum was a key piece to this massive puzzle that is this project,
- Plus, I have consolidated my abilities with Linux-based Operating systems, as well as with Docker and Python as the main technologies used for this project,
- And last but not least, I have learned the utmost importance of writing documentation for the things you do, no matter what they are or the relevance they have in the project. As this project has spanned almost two years, the things I did at the start are very vague in one's memory. So having to write this very document was a huge effort.

Although the results are not great (with a 53% WER), they resemble the potential of this project. We have explained every step followed from the data collection, preprocessing, training up to results obtention, and model assessment. This is not only a success but a fulfillment of the original objective of this project.

However, we need to make clear that there is a long way to go at improving the procedures followed, the scripts used, and the data used as input. And, even knowing that, as of today, the DeepSpeech project is partially discontinued [7], we have laid the basis to creating an STT system more than capable of performing Spanish speech transcription, given the necessary adjustments.

7 Annex I: Complementary files

I provide a link to a OneDrive Cloud Storage folder that contains several files and folders. These contents serve as aid in the understanding of all the items explained in this document and as proof for the tasks mentioned previously.

Inside the folder, I have included a 'README.txt' file that explains the contents in more detail.

`https://unioviedo-my.sharepoint.com/:f:/g/personal/uo264469_uniovi_es/Egs0tHl1WkZLj6XJeD_o_LUBYiJp-Y7equSy7b67ldEBpA?e=wH8WQt`

The files included in such folders are referenced throughout this document using relative paths to the mentioned directory.

References

[1] Graves Alex. *Supervised Sequence Labelling with Recurrent Neural Networks*. 1st ed. Studies in Computational Intelligence 385. Springer-Verlag Berlin Heidelberg, 2012. ISBN: 3642247962,9783642247965.

[2] Fabian Brackhane and Mária Gósy. "Kempelen's Speaking Machine: Experiences with Replicas." In: *Proceedings of the Second International Workshop on the History of Speech Communication Research (HSR)*. Aug. 2017, pp. 25–34.

[3] Fang Chen and Kristiina Jokinen. "Speech Technology". In: *Speech Technology: Theory and Applications*. New York, NY: Springer, 2010, pp. 25–31.

[4] Weibin Chen. "Phoenix : deep speech based automatic speech recognition system for Italian language". MA thesis. ING - Scuola di Ingegneria Industriale e dell'Informazione, 2019.

[5] *¿Cómo funciona el reconocimiento de voz para transcribir un audio?* `https://www.amberscript.com/es/blog/como-funciona-el-reconocimiento-de-voz-para-transcribir-un-audio/`. Accessed: 2022-01-23. 2020.

[6] Computer History Museum (CHM). *Timeline of Computer History*. `https://www.computerhistory.org/timeline/computers/`. Accessed: 2021-02-23. 2021.

[7] Reuben (Mozilla Discourse). *Future of DeepSpeech / STT after recent changes at Mozilla*. `https://discourse.mozilla.org/t/future-of-deepspeech-stt-after-recent-changes-at-mozilla/66191`. Accessed: 2022-01-20. 2020.

[8] Luke Dormehl. *Today in Apple history: Siri debuts on iPhone 4s*. `https://www.cultofmac.com/447783/today-in-apple-history-siri-makes-its-public-debut-on-iphone-4s/`. Accessed: 2021-03-04. 2020.

[9] *Evaluate and improve Custom Speech accuracy*. `https://docs.microsoft.com/es-es/azure/cognitive-services/speech-service/how-to-custom-speech-evaluate-data`. Accessed: 2022-01-23. 2021.

[10] Dan Froomkin. *How the NSA converts spoken words into searchable text*. `https://theintercept.com/2015/05/05/nsa-speech-recognition-snowden-searchable-text/`. Accessed: 2021-03-04. 2015.

[11] Erich Gamma et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. 1st ed. Addison-Wesley Professional, 1994. ISBN: 0201633612. URL: `http://www.amazon.com/Design-Patterns-Elements-Reusable-Object-Oriented/dp/0201633612/ref=ntt_at_ep_dpi_1`.

[12] Awni Hannun et al. *Deep Speech: Scaling up end-to-end speech recognition*. 2014. arXiv: `1412.5567 [cs.CL]`.

[13] IBM. *IBM Shoebox*. `https://www.ibm.com/ibm/history/exhibits/specialprod1/specialprod1_7.html`. Accessed: 2021-03-01. 2021.

[14] Neha Jain and Somya Rastogi. "Speech recognition systems - A comprehensive study of concepts and mechanism". In: *Acta Informatica Malaysia* 3 (Jan. 2019), pp. 01–03. DOI: `10.26480/aim.01.2019.01.03`.

[15] Uday Kamath, John Liu, and James Whitaker. *Deep Learning for NLP and Speech Recognition*. Springer, 2019.

[16] Sarfaraz Masood et al. “Isolated word recognition using neural network”. In: *Proceedings of the 2015 Annual IEEE India Conference (INDICON)*. Dec. 2015, pp. 1–5. DOI: `10.1109/INDICON.2015.7443697`.

[17] Medium. *Speech Recognition Technology: The Past, Present, and Future*. `https://medium.com/swlh/the-past-present-and-future-of-speech-recognition-technology-cf13c179aaf`. Accessed: 2021-03-04. 2018.

[18] Daniel Michelsanti et al. “Deep-learning-based audio-visual speech enhancement in presence of Lombard effect”. In: *Speech Communication* 115 (2019), 38–50. ISSN: 0167-6393. DOI: `10.1016/j.specom.2019.10.006`. URL: `http://dx.doi.org/10.1016/j.specom.2019.10.006`.

[19] Ahmed Moawad. “Speech Recognition System”. In: *Smart Blind Stick Book*. Information Technology Institute, Cairo, Egypt, July 2012. Chap. 2.

[20] Mozilla. *DeepSpeech Playbook*. `https://mozilla.github.io/deepspeech-playbook/`. Accessed: 2022-01-10. 2021.

[21] Mozilla. *DeepSpeech project documentation*. `https://deepspeech.readthedocs.io/en/r0.8/index.html`. 2019.

[22] Felp Roza. *End-to-end learning, the (almost) every purpose ML method*. `https://towardsdatascience.com/e2e-the-every-purpose-ml-method-5d4f20dafee4`. Accessed: 2021-04-17. 2019.

[23] Johan Schalkwyk et al. “Google Search by Voice: A Case Study”. In: *Advances in Speech Recognition: Mobile Environments, Call Centers and Clinics*. 2010, pp. 61–90. URL: `http://dx.doi.org/10.1007/978-1-4419-5951-5_4`.

[24] SONIX. *A short history of speech recognition*. `https://sonix.ai/history-of-speech-recognition`. Accessed: 2021-03-01. 2021.

[25] Bruce T. Lowerre. “The HARPY Speech Recognition System”. Thesis summary. PhD thesis. Pittsburgh, Pennsylvania 15213: Carnegie-Mellon University, Apr. 1976.

[26] V.Suganya V.Ajantha Devi. “An Analysis on Types of Speech Recognition and Algorithms”. In: *International Journal of Computer Science Trends and Technology (IJCST)* 4.2 (2016), pp. 350–355.