Universidade de Évora - Escola de Ciências e Tecnologia

Mestrado em Engenharia Informática

Dissertação

Deep learning for speech to text transcription for the Portuguese language

Eduardo Farófia Medeiros

Orientador(es) | Paulo Miguel Quaresma

Luís Rato

Évora 2023

Universidade de Évora - Escola de Ciências e Tecnologia

Mestrado em Engenharia Informática

Dissertação

Deep learning for speech to text transcription for the Portuguese language

Eduardo Farófia Medeiros

Orientador(es) | Paulo Miguel Quaresma

Luís Rato

Évora 2023

A dissertação foi objeto de apreciação e discussão pública pelo seguinte júri nomeado pelo Diretor da Escola de Ciências e Tecnologia:

Presidente	|	Lígia Maria Ferreira (Universidade de Évora)
Vogais	|	Luís Rato (Universidade de Évora) (Orientador) Miguel José Barão (Universidade de Évora) (Arguente)

Évora 2023

To Yaroslav, João, Francisco, and Danilo, as well as my parents, Dinis and Isabel, who have truly inspired, supported, and believed in me during this adventure.

Acknowledgements

Foremost, I would like to thank Professor Luís Rato for inviting me to join this project on which I would write this dissertation. I would like to thank Professor Luís Rato and Professor Paulo Quaresma for mentoring me throughout this dissertation. To finish off the first batch of acknowledgements, I would like to thank Leonel Corado and Professor Pedro Salgueiro for all the help provided in the countless hours we have spent brainstorming possibilities for improvements and correcting issues during the development of this work.

Secondly, but just as important, I would like to thank Margarida Sampaio for all the psychological support provided along this journey, without which this would not have happened in the same way.

Finally, I want to acknowledge my friends, family and housemates for being supportive throughout the entirety of my academic life and, especially, in the writing of this dissertation.

Contents

List of Figures

List of Tables

Acronyms

AI Artificial Inteligence

ANN Artificial Neural Network

ASR Automatic Speech Recognition

BP Brazilian Portuguese

CNN Convolutional Neural Network

DL Deep Learning

DNN Depp Neural Network

E2E End-to-end

EP European Portuguese

GMM Gaussian Mixture Models

HMM Hidden Markov Models

ML Machine Learning

RNN Recurrent Neural Network

TDNN Time Delay Neural Network

UE Universidade de Évora

WER Word Error Rate

Abstract

Automatic speech recognition (ASR) is the process of transcribing audio recordings into text, *i.e.* to transform speech into the respective sequence of words. This process is also commonly known as speech-to-text. Machine learning (ML), the ability of machines to learn from examples, is one of the most relevant areas of artificial intelligence in today's world. Deep learning is a subset of ML which makes use of Deep Neural Networks, a particular type of Artificial Neural Networks (ANNs), which are intended to mimic human neurons, that possess a large number of layers.

This dissertation reviews the state-of-the-art on automatic speech recognition throughout time, from early systems which used Hidden Markov Models (HMMs) and Gaussian Mixture Models (GMMs) to the most up-to-date end-to-end (E2E) deep neural models. Considering the context of the present work, some deep learning algorithms used in state-of-the-art approaches are explained in additional detail.

The current work aims to develop an ASR system for the European Portuguese language using deep learning. This is achieved by implementing a pipeline composed of stages responsible for data acquisition, data analysis, data pre-processing, model creation and evaluation of results.

With the NVIDIA NeMo framework was possible to implement the QuartzNet15x5 architecture based on 1D time-channel separable convolutions. Following a data-centric methodology, the model developed yielded state-of-the-art Word Error Rate (WER) results of $WER = 0.0503$.

Keywords: Machine Learning, Deep Learning, Deep Neural Networks, Speech To Text, Automatic Speech Recognition, NVIDIA NeMo, GPUs, data-centric, Portuguese language

Sumário

Aprendizagem profunda para transcrição de fala para texto para a Língua Portuguesa

O reconhecimento automático de fala (ASR) é o processo de transcrever gravações de áudio em texto, *i.e.*, transformar a fala na respectiva sequência de palavras. Esse processo também é comumente conhecido como speech-to-text. A aprendizagem de máquina (ML), a capacidade das máquinas de aprenderem através de exemplos, é um dos campos mais relevantes da inteligência artificial no mundo atual. Deep learning é um subconjunto de ML que faz uso de Redes Neurais Profundas, um tipo particular de Redes Neurais Artificiais (ANNs), que se destinam a imitar neurónios humanos, que possuem um grande número de camadas

Esta dissertação faz uma revisão ao estado da arte do reconhecimento automático de fala ao longo do tempo, desde os primeiros sistemas que usavam Hidden Markov Models (HMMs) e Gaussian Mixture Models (GMMs até sistemas end-to-end (E2E) mais recentes que usam modelos neuronais profundos. Considerando o contexto do presente trabalho, alguns algoritmos de aprendizagem profunda usados em abordagens de ponta são explicados mais detalhadamente.

O presente trabalho tem como objetivo desenvolver um sistema ASR para a língua portuguesa europeia utilizando deep learning. Isso é conseguido por meio da implementação de um pipeline composto por etapas responsáveis pela aquisição de dados, análise dos dados, pré-processamento dos dados, criação do modelo e avaliação dos resultados.

Com o framework NVIDIA NeMo foi possível implementar a arquitetura QuartzNet15x5 baseada em convoluções 1D separáveis por canal de tempo. Seguindo uma metodologia centrada em dados, o modelo desenvolvido produziu resultados de taxa de erro de palavra (WER) semelhantes aos de estado da arte de $WER = 0.0503$.

Palavras chave: Aprendizagem de Máquina, Aprendizagem Profunda, Redes Neuronais Profundas, Fala para texto, Reconhecimento Automático de Fala, NVIDIA NeMo, GPUs, abordagens centradas em dados, língua portuguesa

1

Introduction

This introduces the dissertation topic following the subsequent set of contents:

- 1.1 - **Automatic Speech Recognition** - a brief introduction to automatic speech recognition
- 1.2 - **Motivation** - presentation of the motivation and scope of the present dissertation
- 1.3 - **Objectives** - general explanation of the goals of the present work
- 1.4 - **Main contributions** - overview of the main contributions of the present work
- 1.5 - **Structure** - presents the structure of the document

1.1 Automatic Speech Recognition

Automatic speech recognition is the process of putting sounds into text, *i.e.* to transform speech into the respective sequence of words. This process is also commonly known as speech-to-text. In today's world,

most of the writing process has mostly shifted from being done by hand to being done on computers (laptops, tablets or smartphones). Although typing is predominant, dictation has shown to be faster [RWL+16], which opens the opportunity for automatic speech recognition to become the primary way of getting text written. Despite this scenario being a probable future, some work still needs to be developed in the automatic speech field mostly regarding its efficiency in less favourable conditions, *e.g.* noisy environments.

1.2 Motivation

In recent years the field of artificial intelligence (AI), which in its simplest form, is a field, which combines computer science and datasets, to enable problem-solving [Wha], has seen major growth in popularity and in technological advancements. Machine learning (ML), the ability of machines to learn from examples, is one of AI's most predominant fields in today's world. Without realising it ML is present in almost everyone's life, either when using weather forecasts, recommendation systems, for instance, on online shopping or streaming services, or in smart devices to model our routines. Deep learning (DL) is a subset of ML which makes use of Deep Neural Networks, a special type of Artificial Neural Networks (ANNs), which are intended to mimic human neurons, that possess a large number of layers. This large amount of layers allows features to be extracted from the raw input data without any pre-processing needed [Ami21]. Alongside a very rapid growth of available data, DNNs have started to be used in a variety of fields such as computer vision, natural language processing and speech recognition.

In spite of the Portuguese language being one of the most spoken languages in the world, there is not much research developed in the area of speech recognition. The Portuguese language is divided into different variants such as Brazilian and European Portuguese. Among other variants, European Portuguese represents a small fraction of Portuguese speakers hence consequentially having less research developed in this area.

The just-mentioned state-of-the-art methodology and the scarcity of research developed in the area are the main motivation for the current work.

1.3 Objectives

The work presented for this dissertation was developed in the scope of the project "AlticeLabs/Optimized Portuguese Speech To Text" developed in a collaboration between the University of Évora and Altice Labs. Considering the latter topic mentioned in the motivation, the project aimed to develop an automatic speech recognition model using deep learning techniques for the Portuguese language. To achieve such a goal the following objectives were defined:

1. Analysis on the automatic speech recognition state-of-the-art methodologies, frameworks and datasets for the Portuguese language

2. Selection the best fit framework considering the final goal and the infrastructure available

3. Develop experiments based on data pre-processing

4. Test and evaluate models created using the chosen framework and data with different degrees of pre-processing

1.4 Main contributions

The main contributions of the current work are:

- The main output of the present dissertation is the developed model using NVIDIA NeMo. The presented model can be used to perform different tests, *e.g.* with different test sets, or as a starting point for new models.
- The restructured SpeechDat dataset can also be used in future works and different data selection and manifest creation can be achieved using the scripts developed in the current work.
- Under the scope of the current work, a simple web interface and web API were developed in order to provide a more user-friendly interface for audio transcription.

1.5 Structure

The present work is organised into a total of eight chapters. The list below provides a brief description of the topics addressed in each chapter:

- 1 - **Introduction** - introduces the dissertation topic, motivation and objectives
- 2 - **State-Of-The-Art** - provides a review of the literature on automatic speech recognition methodologies
- 3 - **Datasets** - presents an introduction to data on automatic speech recognition and introduces the dataset used in the current work
- 4 - **ASR Deep Learning** - introduces different deep learning algorithms, frameworks and methodologies to automatic speech recognition
- 5 - **Proposed System** - describes the automatic speech recognition system proposed in the current work
- 6 - **System Implementation** - presents the methodology used to implement the proposed system in the previous chapter
- 7 - **Experiments** - describes the infrastructure in which the current work was developed, the experiments and respective results and conclusions
- 8 - **Conclusion and Future work** - presents an overall conclusion of the current dissertation followed by the view on future work

2

State-Of-The-Art

The current Chapter addresses the state-of-the-art in automatic speech recognition and follows the structure presented below.

- 2.1 - **Automatic Speech Recognition** - brief statement on the definition of ASR
- 2.2 - **Approaches to ASR** - study of different approaches used in the development and improvement of ASR systems
- 2.3 - **ASR applications** - examples of today's world applications which use ASR
- 2.4 - **ASR in Portuguese** - review on developed work in this area regarding Portuguese
- 2.5 - **Transfer Learning** - look at a technique used, in ASR and other research fields, for the development of new models using prior developed models as a starting point

2.1 Automatic Speech Recognition

Automatic speech recognition (ASR) is a technology which aims, through the use of a variety of techniques and algorithms, to transcript an acoustic sound, usually a human voice, into text (this task is also known as speech-to-text). This technology can be used for different purposes. As stated by Laurent Besacier *et al.* [BBKS14], different types of speech will produce different types of ASR systems. Thus, speech can be classified into categories such as:

- Spelled speech (with pauses between letters or phonemes)
- Isolated speech (with pauses between words)
- Continuous speech (when a speaker does not make any pauses between words)
- Spontaneous speech (*e.g.* in a human-to-human dialog)
- Highly conversational speech (*e.g.* meetings and discussions of several people)

As said, for different types of speech different types of ASR solutions can be developed, and, thus ASR systems are usually built, according to the context and purpose.

2.2 Approaches to ASR

Approaches to developing ASR systems have been around since the early 50s [DBB52] and have been under continuous development until today. Throughout the years the main focus of ASR systems has shifted between different types of approaches, starting from probabilistic ones, that made use of Markov Models [BJM83] or Hidden Markov Models (HMM) combined with Gaussian Mixture Models (GMM) [ZA, Lee94], to the most up to date end-to-end (E2E) deep neural networks (DNN) [ZXL$^+$, CPS, LLG$^+$19, HDY$^+$12, AAB$^+$] and passing by hybrid approaches [BM94] which made a combination of the previous two. In spite of the state-of-the-art evolution, ASR systems have some components and processes which are present in the majority of the approaches, such as feature extractors, acoustic models and linguistic models.

The following sub-sections present different approaches used to develop ASR systems using different methodologies to implement the previous components.

2.2.1 Probabilistic - Gaussian Mixture Models and Hidden Markov Models

In contrast to E2E systems, probabilistic approaches are often characterised as pipelines since they are composed by a variety of components which depend on the output from other components of the system (Fig 2.1) [GPCB21].

In this type of approach, to be able to transcribe a particular audio, or use it to train the model, the first step is to extract the features from the audio (Fig 2.2). In probabilistic approaches to ASR systems, the first procedure to obtain the audio features is to apply framing and windowing functions to the audio, *i.e.*, respectively, the audio is split into smaller slices, which normally range from 20 ms up to 40 ms, being the most common 25 ms, and its borders are smoothed to reduce the impact of the framing on the statistical properties of the signal. The next step is to apply filters to the raw audio wave. These filters are usually also piped and are used accordingly to desired detail of the audio features. Some examples

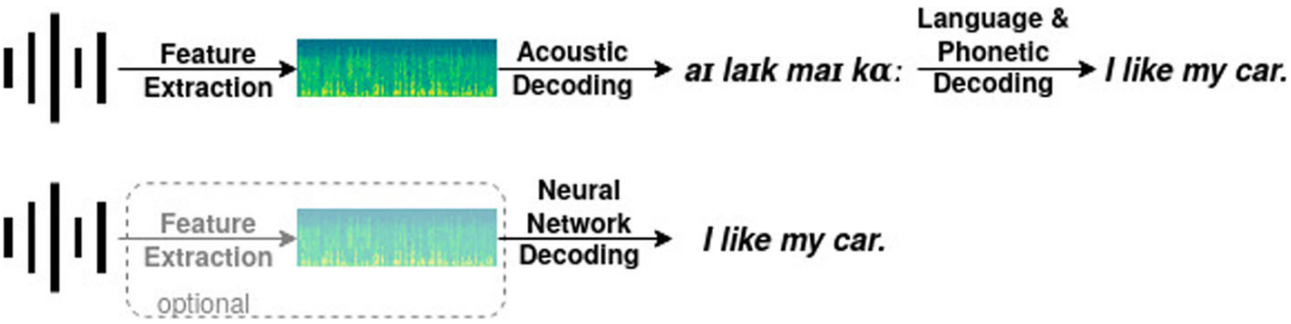

Figure 2.1: Pipeline (top) vs. end-to-end (bottom) ASR ([GPCB21])

of filters are: Fast Fourier Transform (FFT) [HJB84], which changes the signal domain from time to frequency (generates spectrograms), the application of a logarithm function, since humans tend to distinct sounds better when in a logarithmic scale [KLGS17], Mel-filterbanks or Mel-Frequency Spectral Coefficients (MFSC), use triangular filters to reduce the frequency range (usually to the lower part of the spectrum), Mel-Frequency Cepstral Coefficients (MFCCs), reduce the correlation from MFSC coefficients since it might affect the performance of some algorithms [Fay16], and I-vectors, most commonly used when it is intended to retrieve characteristics from the speaker itself [GPCB21].

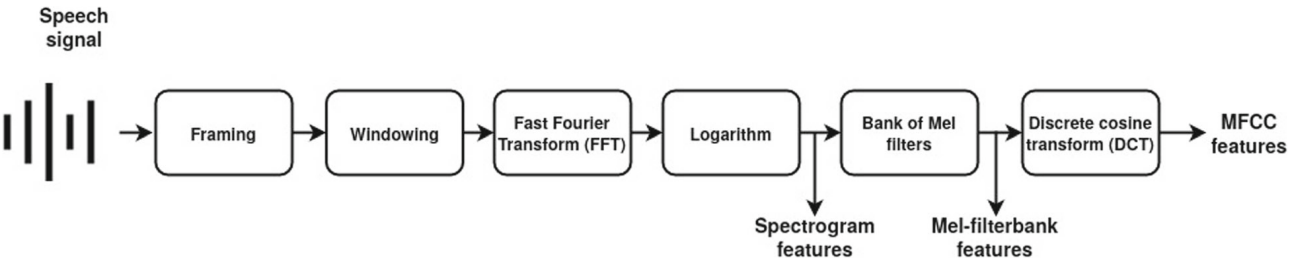

Figure 2.2: Different speech features obtained at different processing stages ([GPCB21])

In this stage, the ASR system already possesses the audio features and can now proceed to the next segment of the pipeline which will be responsible for modelling the base units of the speech, phonemes, *i.e.* by using GMM [ZA] it is able to determine which phoneme is present in the current audio sub-sample.

Following this classification of audio segments into phonemes comes the acoustic model, which is the part of the system responsible for the encoding and decoding of the speech. This model must describe the sequential (dynamic) characteristics of human speech. Such a model can be implemented through the use of an HMM. The HMM is trained with the goal to learn transitions between phonemes, *i.e.* the emission and transition probabilities are calculated using a special type of the expectation–maximization (EM) algorithm, the Baum-Welch algorithm. When instead of training it is pretended to transcribe an audio, the task is to find the best sequence of phonemes. To perform this speech decoding the Viterbi algorithm is used on the previously trained HMM.

Although these components are enough to create a pseudo-ASR system, until this point we are only able to obtain a sequence of phonemes. This output is not human-friendly and doesn't completely fulfil the goal of an ASR system. To complete the system and have a final transcription a phonetic model associates the sequences of phonemes to the respective words. Other models, such as linguistic models, which model the occurrences of sequences of words, can also be used to improve the performance of the system.

2.2.2 End-to-end - Artificial Neural Networks

Recent years have seen a growth in knowledge about the application of ANNs in a variety of research areas, including ASR, computer vision (image processing and recognition), and natural language processing. End-to-end (E2E) methods for ASR systems have increased in popularity due to these developments and the exponential growth in the amount of data that is available.

A common approach to these end-to-end ANNs is to use convolutions. A convolution operation might be interpreted as a filter kernel, *e.g.* a smoothing function, which is applied to another function, *e.g.* an audio signal wave, and the result is the function, the audio signal in the previous example, filtered by the given kernel, which in the example would be the audio signal smoothed by the smoothing function.

A type of ANNs which make use of this operation is Convolutional Neural Networks (CNN). CNNs are frequently used when working in the computer vision domain since they allow to filter the images accordingly to the recognition goal. The same occurs in the audio domain, where instead of an image we have a time series describing the audio signal. In this situation filter kernels are adapted accordingly to features desired to extract from the audio signal which allows CNNs to be used as a replacement for the feature extraction component present in the other types of ASR systems.

Some approaches take advantage of the properties of the convolution operations and use convolutional layers in E2E ANNs to perform the feature extraction step. The layers in these CNNs are representations of determined filters to be applied to the input data, which in this case will be the raw audio [ZXL+]. Among ASR systems that use E2E CNNs some, such as Wav2Letter [CPS], make use of another type of base unit, graphemes, to train the system instead of phonemes. Graphemes are letters that represent a single phoneme, using these as base units removes the need to have a phonetic transcription. Using Wav2Letter's approach as a starting point, Li *et al.* from NVIDIA, developed a new system, Jasper [LLG+19], which uses an even deeper one-dimensional CNN (1D-CNN) (54 layers) and different activation and normalisation functions (ReLU and batch normalisation), which shown to perform better than other function combinations. These modifications showed an improvement in the Word Error Rate (WER) in comparison to other state-of-the-art models.

Creating deeper neural networks comes with the problem of losing parts of the input data in later/deeper layers, and, therefore, this brings the network to stop training furthermore since there is not enough information to update the weights in a better way, also known as vanishing gradient. To address this problem, Huang *et al.*, implemented dense CNNs [HLVDMW17]. In this approach, each of the convolutional layers takes as input the output of each of the preceding layers. Using such a technique enables the dense CNN to reduce the vanishing gradient, making it possible to use the same features in shallower and deeper layers of the network and thus reducing the number of parameters needed for the network.

Another type of ANNs commonly used in ASR systems is Recurrent Neural Networks (RNN). This type of network is often used in problems of the sequential or temporal domain in which a relationship of order between elements can be established, *i.e.* a set of events occur according to a sequence. The input of each layer does not only depend on the output of the previous layer as in other types of ANNs. RNNs possess a so-called "memory" which stores information from previous layers' inputs, which, in combination with the current input will determine the current layer's output. Amodei *et al.* approach, Deep Speech 2, describes an E2E ASR system which makes use of RNNs and is capable of transcribing English and Mandarin [AAB+].

In recent years, Vaswani *et al.* at Google Brain introduced a new architecture (Transformer) based on attention mechanisms. These mechanisms allow modelling sequence dependencies without regard to their distance in the input or output sequences. By using such mechanisms, the Transformer architecture

removes the usage of recurrence and convolutions [VBS+17]. In September 2022, Radford *et al.* at OpenAI developed a neural network based on the Transformer architecture, Whisper [RKX+22]. Whisper is trained on a 680000 hours multilingual and multitask dataset. The diversity and the large dimension of the dataset enable Whisper to achieve robust and accurate speech recognition when compared to humans.

2.2.3 Hybrid - Hidden Markov Models and Artificial Neural Networks

Hybrid approaches to ASR systems are, in short, a mix of probabilistic ones, discussed in Section 2.2.1, and End-To-End approaches, discussed in Section 2.2.2. Hybrid approaches to ASR systems using Artificial Neural Networks (ANN) have been proposed since the mid-90s. Bourlard *et al.* [BM94], described an approach on how ANNs and HMMs can be used together, pointing out some advantages and disadvantages of using this combination of components in a hybrid approach and also stating the roles assigned to the ANNs (feature extraction) and to the HMMs (probabilities emission). Hinton *et al.* [HDY+12] later stated about Bourlard *et al.* that, at the time, neither the computing power nor the algorithms used were powerful enough to train models with performances good enough to reach, or even outperform, the performances achieved by the previous combination of HMMs and GMMs.

Replacing GMMs and audio filtering with ANNs to perform the feature extraction and determination of the probabilities for the HMMs is not an easy task due to the difference in the input data required by the two models. Some approaches try to make the link between GMMs and ANNs, either by using ANNs to generate sub-word probabilities as a predecessor step of GMMs [HES00], or in the opposite way, by using features provided by GMMs to train ANNs. The last method aims to be used as a universal method for combining GMMs with the most commonly used types of ANNs, Deep Neural Networks (DNN) and Time Delay Neural Networks (TDNN) [TKEY].

Instead of using a combination of GMMs and popular types of ANNs for this purpose (DNNs or TDNNs), some researches discarded GMMs and made direct use of ANNs instead, particularly DNNs, in combination with HMMs. Using these types of ANNs to simply replace feature extraction proceedings and GMMs, by adapting the data as needed, has shown to provide better results in various datasets, especially large ones. Some researches improved DNNs performance by using generative models, that were contextless about HMMs states, and then using these results to initialise the DNNs [HDY+12].

2.3 ASR applications

ASR systems have multiple applications in today's world which can vary accordingly to the specific domain, *e.g.* health, industry, telecommunications, and the most common, smart devices and smart assistants. In the context of healthcare, ASR systems can be used to speed up the documentation of medical records, this speed-up is achieved since most of the paperwork can be taken care of quicker by using speech instead of writing. This speed-up leads doctors to have more free time to take care of other patients [Tob05]. In the industrial environment, ASR systems can provide help to workers in assembly lines, *e.g.* to pop up a certain page of an instruction manual while mid-assembly. In telecommunications, these systems are used to speed up customer support. Telecommunication companies usually use ASR systems to provide automated attendants to understand and direct the customer to the type of support needed [Rab97]. The most popular method of interaction with ASR systems is through the use of smart assistants present on so-called smart devices. The most popular smart assistants are developed by tech companies, such as Google (Google Assistant), Amazon (Amazon Alexa) and Apple (Apple Siri). After being prompted with specific keywords, *e.g.* "Ok Google" or "Alexa", these smart assistants make use of ASR systems to transcribe the speech of the user and then execute the pretended task.

Although computationally demanding some smart devices are capable of running ASR models to transcribe audio locally. For devices that run on batteries where power efficiency is key, *e.g.* smartphones and tablets, performing audio transcription tasks locally is computationally expensive. Setting up devices with more powerful hardware to perform such tasks and transcribe audios locally would lead to batteries running out faster (or the need for bigger ones) and heat dissipation issues, thus, instead of running ASR models locally and being able to perform transcriptions offline, these devices usually sacrifice this feature and use remote services to perform transcriptions. Companies such as Facebook, Google and Microsoft provide these services through APIs. These APIs use models based on neural networks but differ in type and in the amount of data used for training as can be seen in Table 2.1.

Company	Architecture	Train hours
Facebook	Encoder-Decoder built with fully connected CNN	1041
Google	LAS with Multi-headed Attention	2025
Microsoft	CNN Encoder and BiLSTM Decoder	12500

Table 2.1: APIs architecture and respective training hours

Sampaio *et al.* [XSPMLCdS$^+$] evaluated these three APIs using two collaborative and public Portuguese datasets, Mozilla Common Voice[1] and the Voxforge[2]. The result of each API over each dataset can be seen in Table 2.2.

Company	MCV WER	Voxfoge WER
Facebook	12.29%	11.44%
Google	12.58%	10.49%
Microsoft	**9.56%**	**7.25%**

Table 2.2: APIs results on Mozilla Common Voice (MCV) Corpus and Voxforge Corpus datasets

Sampaio *et al.* also concluded that the mentioned APIs show similar performances among the used metrics being Microsoft's the one that performed best and also observed that gender influences the performance having the APIs performed better when transcribing male voices.

2.4 ASR in Portuguese

The Portuguese language is one of the most spoken in the world, not due to the size of Portuguese population (10 million), which gives the language its name, but thanks to countries with a much larger number of inhabitants, such as, Brazil (214 million), Angola (33 million) and Mozambique (32 million). Despite speaking the same language the speech varies from country to country and even from region to region, not only in accent but also in vocabulary, *e.g.* the northern region of Portugal has a different accent from the Alentejo region, and words used to describe the same things also tend to vary. The goal of this work is to use European Portuguese (EP), *i.e.* from Portugal, to develop the ASR system.

As already mentioned, EP does not have many speakers when compared to Brazilian Portuguese (BP) or other variants. However, some research has already been developed in the field of ASR with the goal of transcribing EP speech. Pellegrini *et.al.* [PHBDM$^+$13] and Hämäläinen *et al.* [HCC$^+$] aimed to transcribe speech from elder and young people since in these age groups people have more difficulties expressing themselves, therefore the goal was to improve the understatement of their speech through the use of ASR systems. Other research aimed to create a speech recogniser for EP based on a corpus obtained from

[1] `https://commonvoice.mozilla.org/pt`
[2] `http://www.voxforge.org/pt`

broadcast news and newspapers. The AUDIMUS.media [MCNT03] speech recogniser makes use of a hybrid system, a combination of an ANN, in this case, a multilayer perceptron (MLP), which is used to classify the phones given the features extracted using Perceptual Linear Prediction (PLP), log-RelAtiveSpecTrAl (Log-RASTA) and Modulation Spectrogram (MSG) separately, which are then combined and used in an HMM for temporal modeling [MAP+].

In variants of the Portuguese language with a larger amount of speakers, such as Brazilian Portuguese, there is also a lack of work related to the development of ASR systems. This shortage is mostly due to the lack of data quantity, quality or detail on public datasets, or even being public at all, which is much needed especially when creating models based on DNNs. Lima *et al.* [AdLDCA20] provided a list of 24 datasets which contain the Portuguese language alongside some of the available features, such as size, quality, rate, amount of speakers, speaker's age and if it's either public or private. Of the 24 shown, only 6 of the list are public which leads Lima *et al.* to state that the amount of datasets available to build ASR systems for the Portuguese language is acceptable and the types of data are diverse (with noise, different age ranges, medical, commands), but the overall quantity, quality and standardisation of the same are poor. Nevertheless, some research has shown possible to create models for ASR systems for the Portuguese language using reduced amounts of data, as little as 1hr, and achieve some considerable results of WER, 34% [GCO+21]. Other works regarding ASR systems for Portuguese using DNNs worth mentioning are: Gris *et al.* [RG] that makes use of Wav2vec 2.0 and pre-trained models in other languages (then fine-tuned to BP) and achieves an average WER of 12.4% on 7 datasets; Quintanilha *et al.* [MQ17, QNB20] makes use of 4 datasets (3 of which are open), and use models based on DeepSpeech 2 [AAB+] with convolutional and bidirectional recurrent layers, making possible to achieve values of 25.45% of WER.

2.5 Transfer Learning

As stated before, despite having many speakers, Portuguese is a language with a great lack of data and the existing one being of poor quality (explored in section 2.4). This factor is pointed out as one of the most probable causes why the development of ASR systems for Portuguese [GCO+21] and other low-resourced languages is such a difficult task.

Transfer learning (TL) is a process in machine learning (ML) in which different sets of data are used to complement each other, *i.e.* data from a given set is assumed to have enough of the pretended characteristics that denote the goal of the given task, which will help to generalise the data of a second set [GBC16].

This has been shown helpful to develop or improve a variety of systems where performance tends to depend on the amount of available data for training. *E.g.* a system trained to detect a type of animal, *e.g.* cats, can be expanded to detect additional types, lions for instance if given enough examples of the new ones, which usually have much fewer examples than the first ones. As for ASR systems, most rely on the same principles, extracting features from the audio and then determining the sequence of phonemes. These steps are common to most ASR systems and are independent of the language for which they are built. Thus, transfer learning can be used as a means of creating ASR systems for languages with limited amounts of data. Such can be achieved by transferring the knowledge of the previously mentioned steps from models developed for languages with a greater amount of accessible data, such as the English language. The already trained steps are then tuned for languages with fewer amounts of available data, such as the Portuguese language.

Transferring knowledge on ANNs-based systems is the equivalent of reusing previously trained layers from other models. This is done by reusing the already calculated weights to initialise determined layers of new models followed by training the remaining layers. The reused layers of the new model can behave differently

accordingly to what is desired, they can either be fixed, which means the weights brought from the previous model won't change or be flexible, in which case the weights will be able to change a re-calibrate accordingly to the new data [BKD21]. As for the remaining layers, the weights are randomly initialised as in a normal ANN.

Working with weights from pre-trained models to initiate layers for new models comes with some challenges. It is not easy to decide in which layers to split the network and which to reuse in new models due to some layers being very fragile when it comes to performing this splitting of the network. Transferring layers closer to the output is also a challenge since they may be too fitted to the original model, which makes it more difficult to adapt to the new one. These challenges vary in dominance accordingly to where the knowledge is being transferred from, retrieving weights from deeper or shallower layers determines how much each of these challenges alters the transfer process. The dissimilarity among the tasks used as starting and ending points of this process, *i.e.* how distant the original and target are from each other, is also a factor that influences the performance of the transfer process. In spite of these challenges and concerns about specialisation versus generalisation of layers, it has been shown that transferring knowledge among ANNs, even if from reasonably distant tasks, outperforms models initialised with random weights [YCBL].

3

Datasets

Data is a crucial component in creating ASR systems because it is strongly necessary for the learning stage. In this Chapter we find:

- 3.1 - **Datasets for ASR** - a brief introduction about datasets in ASR
- 3.3 - **Multilingual LibriSpeech** - presentation of the the open-source dataset based on LibriSpeech
- 3.4 - **SpeechDat** - presentation of a proprietary dataset made available for the development of the present work
- 3.2 - **LibriSpeech** - presentation of a open-source English dataset

3.1 Datasets for ASR

The performance of ASR systems doesn't only rely on the type of algorithms used, probabilistic or end-to-end, it also depends on the quality and quantity of data available to train them. Some languages, such as

English, are widely present in the majority of the ASR applications of today's world, mostly smart assistants, and have various research works in the field of ASR, *e.g.* [SKSKA+17]. A larger number of speakers provides larger amounts of data to be available for these studies to be developed. In spite of some languages having a wide range of data sources available others lack it, which is the case of the Portuguese language. There are some sources of audio and the respective transcriptions available for Portuguese, but the quantity and quality are not usually good enough to create ASR systems with acceptable performance. These sources tend to fail in one or more important aspects of the dataset, either by having audio or transcriptions with poor quality, low quantity of data or lack of structure standardisation. Lack of structure standardisation is also an important aspect since it increases the difficulty of creating automatic data processors for different data sources. There are a few different paths which can be used to gather data to develop an ASR system:

- private entities - through an agreement provide private datasets
- open access datasets - dataset available to use
- crowdsourcing - large groups of people donate data
- creating a dataset from scratch - using available data from different sources of audios and transcriptions, *e.g.* audiobooks and respective transcriptions

3.2 LibriSpeech

As the amount of content available online increases, more data can be collected for research purposes. Such data can be used to construct machine learning models for computer vision, natural language processing, and automatic speech recognition, among others. When well organised and structured, even if from different sources, these data can be easily retrieved by computers and used for the creation of datasets.

Taking advantage of this trend, the LibriSpeech[1] dataset was built using data from the LibriVox[2] audiobook catalogue. Composed of 1000 hours of audio recordings of English speech, with a sampling rate of 16 kHz, LibriSpeech was created with the purpose of being used to build and test ASR systems [PCPK15]. The total 1000 hours of audio recordings are divided into three partitions. One with 100, and another with 360 hours of audio recordings, both containing audio with higher quality recordings and accents closer to US English, and a third partition composed of approximately 500 hours of audio recordings. Speakers were characterised as "clean" or "other" according to the WER of their transcripts obtained from a model trained on Wall Street Journal data. 40 random "clean" speakers, 20 female and 20 male, were assigned to a development set and the same process was repeated for a test set. The remaining "clean" speakers were randomly assigned to train sets, one for each of the previously mentioned partitions of 100 and 360 hours of audio recordings. Speakers labelled as "other", were divided into development, test and a train set with 500 hours of audio recordings.

In the present work, the 100 hours "clean" train set was used as the control dataset. This control was meant to verify the performance of the transfer learning process, *i.e.*, due to the disparity in the amount of data, to verify if the models pre-trained in English were properly learning/adjusting to the Portuguese data. The 100 hours set contains transcripts with a total of 990101 words, from which 33798 are unique words. The average amount of words per sentence is 34.69.

[1] `https:///www.openslr.org/12`
[2] `https://librivox.org`

3.3 Multilingual LibriSpeech

OpenSLR[3] is a popular public bank of speech and language resources. This bank provides the ability for individual contributors and organisations to publicly share resources and make them available for download. In this bank we can find the LibriSpeech and the Multilingual LibriSpeech[4] (MLS) datasets. LibriSpeech is a dataset composed of 1000 hours of English speech audio recordings, based on data from the LibriVox audiobook catalogue, and was created with the purpose of being used to build and test ASR systems [PCPK15]. LibriSpeech was also used as the starting point for the creation of the other mentioned dataset, the Multilingual LibriSpeech.

MLS is an extension of LibriSpeech in which the amount of English speech available is increased to 44.5K hours of audio recordings and seven other languages are added with a total of 6K hours of audio recordings. The new languages are German, Dutch, Spanish, French, Portuguese, Italian and Polish [PXS+20]. Since LibriSpeech is used as starting point, MLS also uses audiobooks available at LibriVox. After obtained, the audio from the audiobooks was segmented into 10-20 seconds segments at the longest silent part of that interval, or, if there was no silence chunk in this interval, at the 20 second mark. Making use of the previous segmentation's acoustic models, pseudo labels were then generated for the audio segments. The audiobooks' data is then downloaded, parsed and normalised, *i.e.* some characters, like punctuations, emojis and escape symbols, are removed. The original text is then split into overlapping documents. These are retrieved if their content is one of the best matches for the pseudo labels of a certain audio segment, but are only kept if the respective WER is lower than 40%. Further processing regarding numbers, hyphens and apostrophes also took place. Pseudo labels, which matched the alignment of a certain number on the book text, were used to replace numbers, and heuristics were used to determine where to remove or keep hyphens and apostrophes in unexpected places.

MLS provides the dataset split into three sets, train, development and test in which there is no speaker overlap. For the last two sets, it is also guaranteed that the gender and duration of the speaker are balanced. These splits only include data from books with good metadata, *i.e.* there is no missing data such as title or speakers' or authors' information. Audios are also ensured to unambiguously contain only one speaker. This division into sets starts by ordering the speakers by the total duration of their speech. The speakers whose total duration of readings is lower than a threshold are assigned to the training set, while the remaining are grouped by the shortest duration of the speech equally in each gender and are then divided equally by the development and test sets. Every remaining speaker is assigned to training.

Regarding the English language, the one which MLS extended from LibriSpeech, and the Portuguese language, the one used in the present work, it is presented, in Table 3.1, the number of hours of audio recordings originally present in LibriVox and then on MLS after the just stated construction steps took place. With respect to Portuguese, MLS contains a large number of hours ($\simeq$167 hours of Brazilian and $\simeq$1 hour of European Portuguese audio recordings) when compared with the majority of those (shown in hours) stated by Lima *et al.* [AdLDCA20]. These hours correspond to a total of 1321326 words, of which 77292 are unique, and an average of 33.68 words per transcription.

3.4 SpeechDat

The SpeechDat European Project[5] was developed between March 1st of 1996 and February 28th of 1998 with the objective of providing speech resources to stimulate research and development of automated

[3] https:///www.openslr.org/
[4] https://www.openslr.org/94
[5] https://cordis.europa.eu/project/id/LE24001

Language	LibriVox	MLS			
		Train	Development	Test	Total
English	71506.78	44659.74	15.75	15.55	44691.04
Portuguese	284.59	160.96	3.64	3.74	168.34

Table 3.1: Hours of audio recordings of English and Portuguese present in LibriVox audiobooks and MLS dataset

services such as speech recognisers. SpeechDat databases covered all 11 official languages of the European Union. Regarding Portuguese, the database was collected by Portugal Telecom, now named Altice Portugal[6], in collaboration with INESC and INESCTEL.

The current work was developed as a collaboration between Universidade de Évora and Altice Labs. Being Altice Labs part of Altice Portugal, owner of the SpeechDat dataset, considering the scope of the project this private dataset was made available for the development of the current study regarding ASR.

Data collection for the Portuguese database of the SpeechDat project was made by INESCTEL. Regarding this task 4027 Portugal Telecom employees were selected as speakers for this data retrieval. Since the company employees are widely spread geographically, they are guaranteed a good representation of regional accents. These speakers were given prompt sheets to follow during the audio recordings. Some of these sheets were specially designated for speakers who volunteered to perform recordings from public phone booths.

Accordingly to the dataset's documentation, the data retrieved from all recordings, the speech signals and respective metadata, separate files for the speech signal and the respective headers. The audio files were encoded using A-LAW [7] (an algorithm used for encoding audio signals, in particular, voice encoding) with a sampling rate of 8 kHz 8-bit, which was accompanied by an ASCII label file containing rows regarding the speech file:

- identification
- session
- recording conditions
- speaker
- file

Each label file contained an assessment code regarding the quality of the respective audio. The possible values for the code are the following:

- **OK** - clean audios and ready to be used
- **NOISE** - audios with some background noise
- **GARBAGE** - empty audios, missing transcriptions, only background noise, noise produced by others
- **OTHER** - audios containing disfluencies, hesitations, stuttering or unintelligible speech

[6] https://www.telecom.pt/
[7] https://en.wikipedia.org/wiki/A-law_algorithm

Quality Label	Hours
OK	152.99
NOISE	30.82
GARBAGE	1.04
OTHER	0.34
NO_PTO	0.90
TOTAL	186.09

Table 3.2: Hours of audio recordings for each audio quality label

The project's documentation didn't possess the Portuguese database size, thus, the number of hours of audio recordings of each audio quality label (assessment code) was calculated and summed to a total. During this process it was noticed that some audio files didn't possess the respective label file, therefore, their duration wasn't summed to the total. To address this problem, a pseudo audio quality label ("NO_PTO") was created to which these audios were assigned to. Table 3.2 presents the number of hours of audio recordings of each audio quality label and the total hours of audio recordings of the SpeechDat dataset. The 186 total hours of audio recordings represent a lexicon of approximately 15000 different words.

When comparing Table 3.1 and Table 3.2, it can be concluded that the SpeechDat dataset is slightly larger than the Multilingual LibriSpeech by $\simeq$18 hours. This reinforces the scarceness statement of speech data concerning the Portuguese language.

4

ASR Deep Learning

Throughout time deep learning has varied in popularity and has been used in different areas. Deep learning's popularity is described in three waves. The first wave surged in the 1940s-1960s with cybernetics, from 1980-1995 a second wave rose with the usage of neural networks and the third and current wave began circa 2006 [GBC16].

In the current Chapter we find:

- 4.1 - **Artificial Neural Networks** - an introduction to artificial neural networks
- 4.2 - **Deep Learning** - defining Deep Learning
- 4.3 - **DNNs in ASR** - a look on the usage of Deep Neural Networks to perform Automatic Speech Recognition
- 4.4 - **Deep learning frameworks** - an overview of frameworks commonly used to develop state-of-the-art deep neural models
- 4.5 - **Data-centric** - a view over a methodology to develop deep neural models based on data processing

4.1 Artificial Neural Networks

Artificial neural networks are machine learning models inspired by the human brain and have as base units neurons/nodes (illustrated in Figure 4.1) that are based on human neural neurons as well.

Each neuron of the network generates an output value which can be described by Equation 4.1. The equation can be explained as follows: each neuron in the neural network receives as input the weighted outputs of the N neurons connected to it. These inputs are summed to bias (b) and passed through an activation function f. The activation function determines the output value of the neuron (y). This value determines the impact of the neuron on the network or whether it is used at all. The simplest example of an artificial neural network is a network with a single neuron generally called a perceptron or linear classifier. This is exemplified in Figure 4.1.

$$y = f(b + \sum_{i=1}^{N} x_i w_i) \tag{4.1}$$

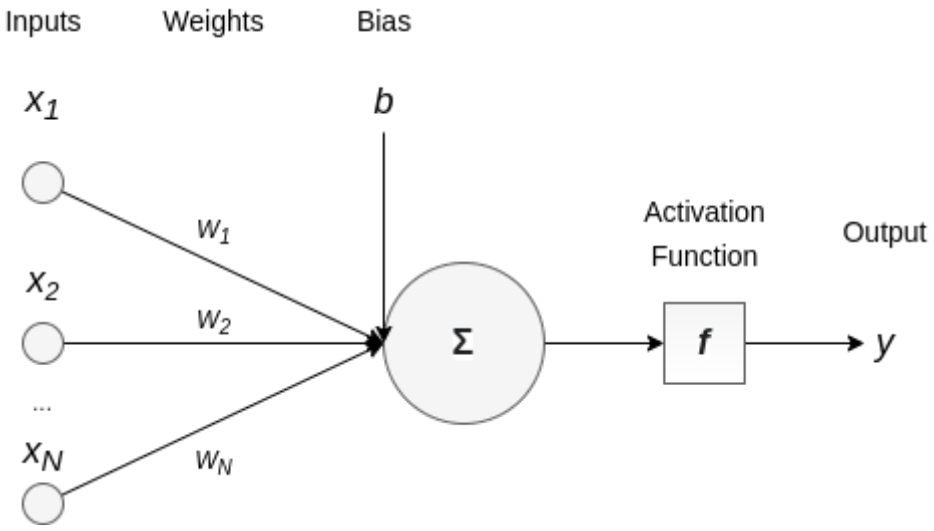

Figure 4.1: Artificial neural network neuron example

In ANNs neurons are grouped together to create layers. Accordingly to their role, layers are normally classified into one of three different types: input, hidden or output layers. Using weighted edges, these layers are connected and thus form a network that resembles the neural networks of humans and other animals [Wan03]. An example of a general ANN can be seen in Figure 4.2.

Differentiation is an essential subject when tuning machine learning models. Model tuning is driven by the distance between the true value of a training instance (y) and the output generated by the model ($\hat{y}$) for that same instance. This distance is usually called error, or loss, and is determined by a loss function that is selected according to the model type and/or problem objective. The use of gradient methods allows the convergence of the model's loss function to a minimum, which means that the model will be optimised in order to generate the smallest distance between y and $\hat{y}$.

The role of the activation function (f) in a neural network is to introduce a non-linear element to the network. If no activation function, or a linear activation function, is used, only linear transformations will be performed across the entire ANN, therefore it could be reduced to a simple linear regression. To

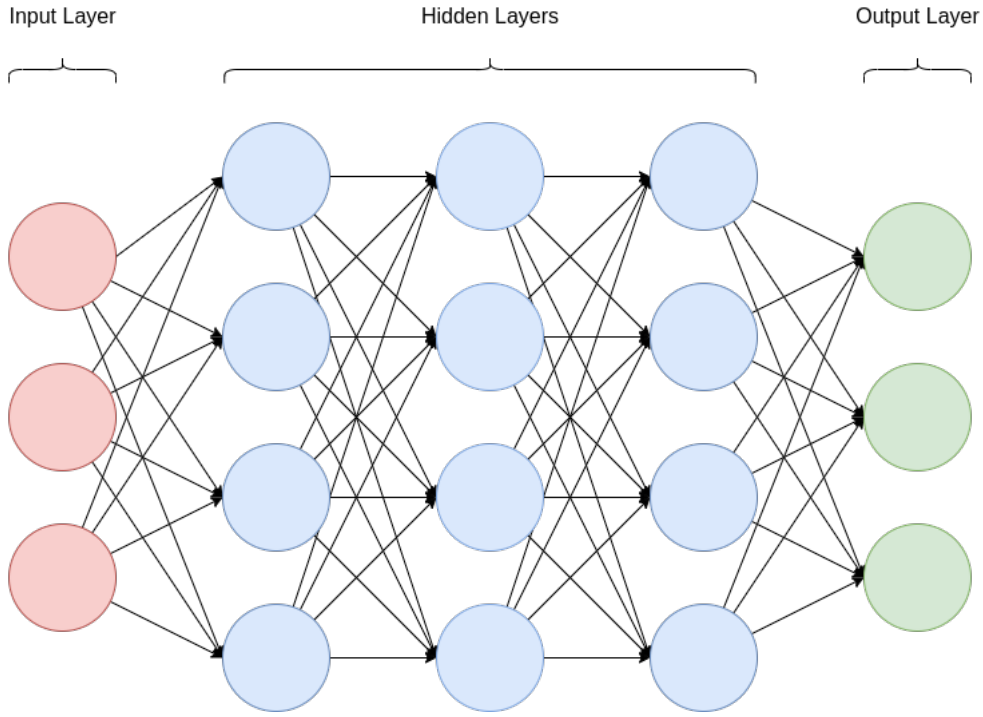

Figure 4.2: Artificial neural network example

overcome this problem, ANNs use non-linear functions instead which enables backpropagation and layers to become non-linear combinations of their inputs. Some examples of these functions are:

- Sigmoid - given by Equation 4.2 - logistic function that varies from 0 to 1
- Hyperbolic tangent - given by Equation 4.3 - logistic function that varies from 1 to -1
- Rectified Linear Unit (ReLU) - given by Equation 4.4 - linear if the value is greater than 0, else 0

$$sig(x) = \frac{1}{1 + e^{-x}} \tag{4.2}$$

$$tanh(x) = \frac{2}{1 + e^{-2x}} - 1 \tag{4.3}$$

$$ReLU(x) = max(0, x) \tag{4.4}$$

Besides the base structure, neurons and layers, ANNs commonly have additional mechanisms, from data pre-processors and augmentation to optimisers and schedulers.

Data pre-processors convert the raw input data into data representations accepted by the network, *e.g.* raw waveforms converted to signal features. While data pre-processors modify existing data, data augmentation methods generate new data by applying transformations to the existing dataset. Data augmentation methods vary according to the problem's domain, *e.g.* in the computer vision domain, image colours, scale or orientation are modified to generate new instances of data.

As stated, ANNs use gradient methods to perform backpropagation and update the weighted edges to reduce the loss value, *i.e.* to reduce the distance between y and $\hat{y}$. Optimisers are the mechanisms responsible for such tasks. ANNs' weights are updated based on the learning rate parameter whose value is determined by a chosen scheduler. Schedulers can range from using a constant learning rate to a varying learning rate which is updated in accordance with a chosen method, *e.g.* Cosine Annealing.

4.2 Deep Learning

Deep learning is a subfield of machine learning with several definitions accepted by the scientific community. Some works, such as [ZYL+18], have put together some common definitions and developed unified ones. Considering some of the mentioned definitions in the previously cited work, deep learning can be defined as a class of machine learning algorithms that, by using deep neural networks (DNNs), ANNs with multiple hidden layers, can model various levels of data representations. DNNs can model different levels of data abstraction, from higher and more abstract levels to lower ones such as raw sound signals or images. This variety of data representation can be achieved due to a large number of hidden layers that apply several transformations and feature extraction methods to the raw data.

Automatic speech recognition systems have evolved over time and passed by different types of approaches as seen in Chapter 2. Deep learning has seen its first appearance in automatic speech recognition systems after 2009 with restricted Boltzmann machines (RBMs) to initialise a DNN whose role was to predict probabilities for HMM states. At the time this approach helped to decrease the phoneme error rate from 26% to 20.7%. A few years forward, automatic speech recognisers based on DNNs, instead of using pre-trained RBMs, started using other methods, that, by eliminating some neurons, decreased the network's over-fitting, which improves generalisation [GBC16]. Examples of the previously mentioned methods are:

1. the use of rectified linear units activation function (ReLU) instead of logistic functions
2. the dropout technique

The first mentioned method, the ReLU activation function, has become the most used activation function in hidden layers, and, according to Zeiler *et al.* [ZRM+13], DNNs using ReLU have shown improvements over the ones using logistic functions, such as:

- being easier to optimise
- faster to converge
- improved generalisation
- faster computation

The second method, the dropout technique, is used to improve the generalisation error associated with large networks, such as DNNs, and decrease their over-fitting. Dropout accomplishes doing so by randomly omitting a fraction of the hidden units in all layers [DSH13].

4.3 DNNs in ASR

Artificial Neural Networks have a wide variety of architectures whose use varies according to the scope and objective of the problem. By selecting a specific class of problems the diversity of architectures decreases.

However, as in the case of automatic speech recognition, there continues to be more than one type of architecture commonly used to solve this problem.

The most notable approaches to state-of-the-art automatic speech recognition systems are largely based on three types of deep neural networks, being them:

- Convolutional Neural Networks (CNNs)
- Recurrent Neural Networks (RNNs)
- Time-Delay Neural Networks (TDNNs)

Convolutional Neural Networks

In contrast to other types of DNNs convolutional neural networks are ANNs where the shallower fully connected hidden layers are replaced with convolutional and pooling layers. Convolutional layers perform convolution operations which is an element-wise matrix multiplication between the input and a kernel (a small matrix of weights). Pooling layers are used to reduce dimensionality. These operate similarly to convolution layers, a window slides over the input and returns the maximum, average or another operation over the current part of the input. This combination of layers allows the network to apply a set of filters to the input. The type of input used in these layers varies accordingly to the domain of the problem being tackled. In the computer vision domain images are used as input to the network [LHB04, LGTB97], but in the speech recognition domain, the input can range from, 1D or 2D feature maps [AHMJ+14] to raw speech signals [PDC15]. Filters not only vary according to the problem's domain but also change throughout the network. Lower layers filter more general features, such as *e.g.* edges, and deeper layers are meant to detect more distinct features, such as *e.g.* specific objects. In the speech domain filters are applied to the speech features which can be represented as an "image", a spectrogram. The spectrogram of a sound represents the amplitude of the sound as a function of frequency and time. Frequencies are important to represent sound features since different combinations of frequencies characterise different pitches and timbres. According to the representation used in these feature maps, the filters used are either one or two dimensions (1D or 2D). Through an analogy with the computer vision domain, filters applied to sounds will get more complex as the depth of the network increases, similarly to images, these will range from simple filters, *e.g.* edges and phonemes, to more complex ones, *e.g.* objects and syllables. To be able to better model the data CNNs stack several convolutional and pooling layers, *i.e.* multiple filters, which increases the depth of the network [SMKR13] and allows for the extraction of more, and more complex, features from the data.

Recurrent Neural Networks

Recurrent Neural Netowrks (RNNs) are a type of Artificial Neural Networks commonly used with sequential data such as language translation and speech. RNNs perform well with sequential data since predictions are based on previous ones, hence the recurrence. Although the current state prediction is based on the predictions of previous states, due to the vanishing gradient, RNNs memory is short and only the latter predictions truly impact the prediction for the current state. Some types of RNNs, such as Long Short-Term Memory networks (LSTMs) [HS97], overcome this issue by implementing gating mechanisms. For instance, LSTMs implement forgetting, ignoring and selection gates. Such mechanisms allow networks to carry knowledge from shallower layers, *i.e.*, from early parts of the sequence, to deeper layers. Consequently, the network can determine which parts of the sequence to memorise for later use thus solving the short-memory problem. RNNs use sequential data from previous stages to predict the current stage output but

in some domains, the input can be processed all at once. In speech recognition, for example, the input is processed at once, hence it is possible to use information from both previous and future stages to make a prediction for the current stage. Bidirectional RNNs (BRNNs) process audio transcriptions in both forward and backward directions to generate a prediction for the current stage. BRNNs can also be improved by using gating mechanisms such as Bidirectional LSTMs [GMH13, GS05]. Deep RNNs are described as a stacking of multiple hidden recurrent layers. Like the ones just mentioned, these deep networks vary depending on the type of hidden units used, "vanilla RNN", LSTM, Bidirectional or Bidirectional LSTM. Deep RNNs are used in end-to-end speech recognition to directly map acoustic features to phonetic sequences [GMH13].

Time-Delay Neural Networks

The speech recognition task aims to transcribe an audio signal into the respective transcription. Time is a very important factor when performing speech recognition since the relationship between the occurrence of phonemes and the time in which they occur greatly impacts the final transcription. RNNs have been shown to be very effective for modelling sequences, such as speech. Time-Delay Neural Networks (TDNNs) are another type of ANNs which are also used to model data according to the context, which, in the case of speech, are the previous time steps. TDNNs base units receive J inputs which are passed by N delays (D). Besides being passed by the delays, each input is also passed without any delay applied to it. These properties enable the TDNN unit to relate and compare present and prior time steps of the speech signal [WHH$^+$89]. TDNNs were used as part of the base of NVIDIA's Japser architecture [LLG$^+$19] which in turn was the foundation for the QuartzNet architecture [KBG$^+$19] used in the current work as described in Section 5.3.2.

4.3.1 Connectionist Temporal Classification

The core concept of automatic speech recognition is to transcribe audio recordings into the correct transcription. Although it may seem like a straightforward task, training a speech recogniser is trickier since it is unknown which characters in the transcript correspond to each audio segment. If not provided, these alignments must be calculated by hand. This task is difficult and time-consuming, especially when working with large datasets containing hundreds or thousands of hours of audio recordings. This labelling can also be accomplished by using algorithms that provide the alignment between the audio signal and the transcriptions.

A possible algorithm is to assert that one output character corresponds to N samples of the input. Although it could be a good approach to some problems, it isn't for speech recognition. This algorithm isn't fit for speech recognition due to a variety of challenges carried out by this task. Both the length of the input and the output might vary which will lead to the 1 to N rule not working in every case. A possible factor for this length variation is the speech rate being different in each person. The speech rate influences the amount of distinct information present in a specific time frame. Lower/slower rates carry less distinct information in the same time interval than higher/faster rates. *E.g.*, if two distinct people say "hello" but at different rates, "heeellooo" (slower) and "heello" (faster), in the same time step, a sample from the lower rate might only contain information to represent "ee" while the higher rate might contain enough information to represent "el". This variation in lengths would lead to character/sound misalignment from person to person if using the 1 to N alignment algorithm. The same problem occurs in handwriting recognition since the space between each character also varies from person to person.

The Connectionist Temporal Classification (CTC) algorithm was designed to overcome these challenges

and provide the alignment between an input and an output sequence. CTC generates alignments between an input sequence (X) and the output labels (Y) by calculating the probability distribution over all possible values of Y. At each time step, for each audio recording segment (x_t), CTC determines the probability of each of the output labels (a_t), which includes the output alphabet and an additional special blank symbol. The CTC alignments provide a direct path to get from probabilities at each time step to the likelihood of an output sequence. Using Equation 4.5, the CTC algorithm is able to calculate $p(Y|X)$, *i.e.* the likelihood of a particular output sequence. This value is obtained by summing over all the probabilities of all the possible alignments between an audio recording X of size T, (x_1, x_2, ..., x_T) and the respective output labels (a_1, a_2, ..., a_T) with the transcript (true output label sequence) Y, (y_1, y_2, ..., y_N), where $N <= T$. We should note that, in general, there are several possible sequences A for each output sequence Y.

$$p(Y|X) = \sum_{A} \prod_{t=1}^{T} p_t(a_t|X) \tag{4.5}$$

This probability distribution can either be used to calculate the probability of a certain output sequence or to infer a likely output label sequence [Han17].

Probabilities generated by CTC can be used as a loss function to train models. Generally, calculating $p(Y|X)$ is very computationally demanding due to the large number of possible alignments that can generate the real output labelling sequence. The computational cost can be lowered by solving this problem using a dynamic programming algorithm. The main concept is that the sum over alignments for a particular output labelling can be transformed into an iterative sum over the alignments of prefixes of that labelling. If two alignments achieve the same output at the same step, they can be combined into one from that step onward [GCF$^+$06, Han17]. CTC loss function, $p(Y|X)$ is composed of sums and products of the output probabilities of each time step, hence, it is differentiable with regard to them. This allows us to analytically determine the gradient of the loss function for the non-normalized output probabilities and then proceed with backpropagation [Han17].

At each time step, transitions can occur between each of the possible output labels, from an output label to the blank symbol or from the blank symbol to an output label. CTC uses the blank symbol (ϵ) to separate labels which are not meant to be collapsed together, *e.g.* a possible output sequence for the word "hello" could be $\texttt{hheee}\epsilon\texttt{ll}\epsilon\texttt{lll}\epsilon\texttt{oo}\epsilon$. To generate "hello" from the previous alignment, CTC first merges sequences of repeated labels into one, thus getting $\texttt{he}\epsilon\texttt{l}\epsilon\texttt{l}\epsilon\texttt{o}\epsilon$, followed by the removal of the blank symbols, which leads to the final alignment `hello`. Without using an extra blank symbol, transcripts with repeated labels, such as the example "hello", couldn't be generated since all the "l" labels would be merged into a single one given that they would appear sequentially.

Given an already trained model, one potential approach to inferring a likely output sequence could be a greedy strategy, such as selecting the higher probability at each time step. Such an approach assumes that selecting the highest probability at each time step achieves the most probable sequence. A better approach is to use a beam search. The beam search has beam width of size B which can be interpreted as the number of sequences explored in each time step. The sequences to be expanded/explored in each time step are the B sequences with higher probabilities from the previous time step. Instead of maintaining a list of B alignments in the beam, the search is optimised by storing the output prefixes after collapsing repeated labels and eliminating blank letters [GCF$^+$06, Han17, Ban19][1]. The accuracy of the inference process significantly improves when a language model is incorporated into the previous approach.

[1]Accessed 13/10/2022 `https://sid2697.github.io/Blog_Sid/algorithm/2019/11/04/Beam-search.html`

4.3.2 Speech features in DL ASR

Speech and other audio signals are frequently represented in the time domain as amplitude waves. Despite some information about the signal being provided by this representation, phonological features cannot be easily extracted. To obtain additional information about an audio signal, waveforms can be changed such that the audio signals can be defined by a set of parameters/features [ST13].

Speech is a *quasi*-stationary signal when considering very small time frames, usually from 5 ms to 100 ms. Slight changes in these frames represent different sounds being spoken. Audio features, determined over these small frames, can be grouped into different categories, such as linear predictive coefficients (LPC) and Mel-Frequency Cepstral Coefficients (MFCCs).

Feature extraction aims to simulate the process carried out by the human cochlea, which performs a *quasi*-frequency analysis on a nonlinear scale. This scale is approximately linear until 1000 Hz, where the human achieves better sound distinction, and approximately logarithmic afterwards [GPCB21]. It is possible to reduce speech variability by representing a speech signal through a set of features, *i.e.* a parametric representation of the speech waveform. Speech variability occurs due to different people speaking in different manners causing variance in speech base characteristics such as pitch, amplitude and frequency.

The LPC technique's key concept is based on determining the current speech sample based on a linear combination of previous ones. At each frame of speech, usually 20 ms, LPC calculates a set of coefficients using the Levinson-Durbin recursion algorithm. Due to high variance, these coefficients are then transformed into a set of parameters known as cepstral coefficients which can be used in speech analysis.

The process of obtaining Mel-Frequency Cepstral Coefficients (MFCCs) is a feature extraction method frequently used in today's ASR systems. Obtaining these coefficients is achieved through a set of methods assembled in a pipeline as shown in Figure 4.3. As in other feature extraction methods, making use of the *quasi*-stationary speech property, the initial step in computing MFCCs is to frame the speech signal with a size of $\approx$25 ms. A windowing function is then applied to the audio frame to eliminate sudden transitions in its boundaries. A Fast Fourier Transform (FFT) is applied to convert the signal waveform from the time to the frequency domain. Only the absolute value is retained from the FFT operation output, which is then utilised to generate the signal's spectrogram. Spectrograms represent the amplitude of a collection of frequencies over time. This representation makes phonological information about the speech more accessible to be used for speech analysis or recognition.

The human ear perceives sound more discriminatively at lower frequencies and less discriminatively at higher frequencies. This implies that people distinguish sounds better at lower frequencies, such as 500-1000 Hz, than at higher frequencies, such as 15000 Hz. Stevens, Volkmann, and Newman created the mel scale with the goal of replicating the perception of sound by the human ear [SVN05]. This scale converts hertz into mels using Equation 4.6 such that equal distances in pitch sounded equally distant to the listener[2].

$$m = 2595 \log_{10}\left(1 + \frac{f}{700}\right) \tag{4.6}$$

Mel-filterbanks features, or Mel-Frequency Spectral Coefficients (MFSCs), are applied to the spectrogram obtained from the FFT in the prior step. The filterbanks, shown in Figure 4.4, are formed of triangular filters that aim to narrow the range of the original frequencies to the lower part of the spectrum by transforming them using the mel scale [Fay16, GPCB21]. Figure 4.5 displays an example of Mel-filterbanks features or Mel-Frequency Spectral Coefficients (MFSCs).

[2]Accessed 27/10/2022 `https://medium.com/analytics-vidhya/understanding-the-mel-spectrogram-fca2afa2ce53`

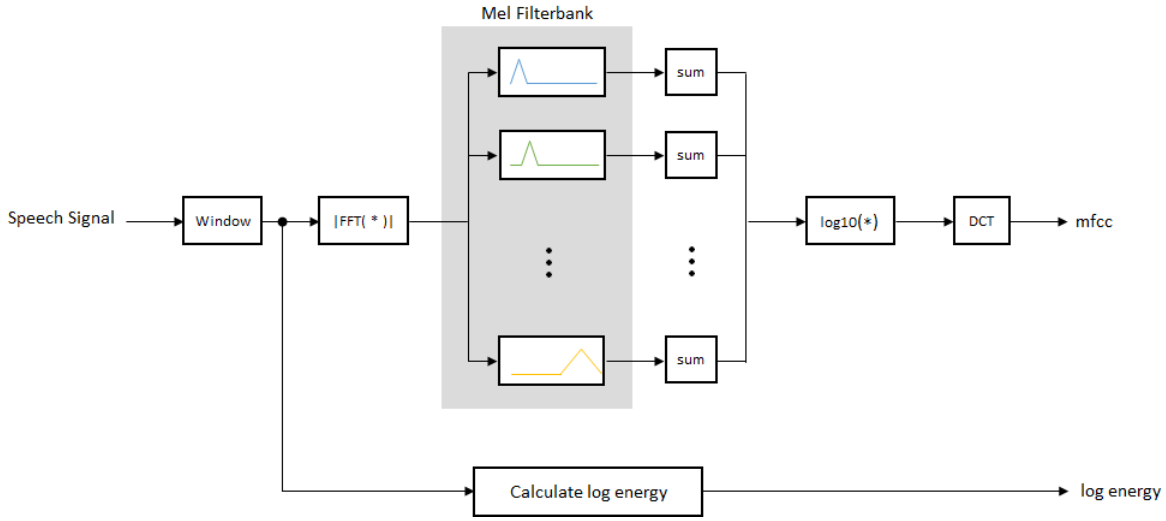

Figure 4.3: MFCCs calculation pipeline [Spe]

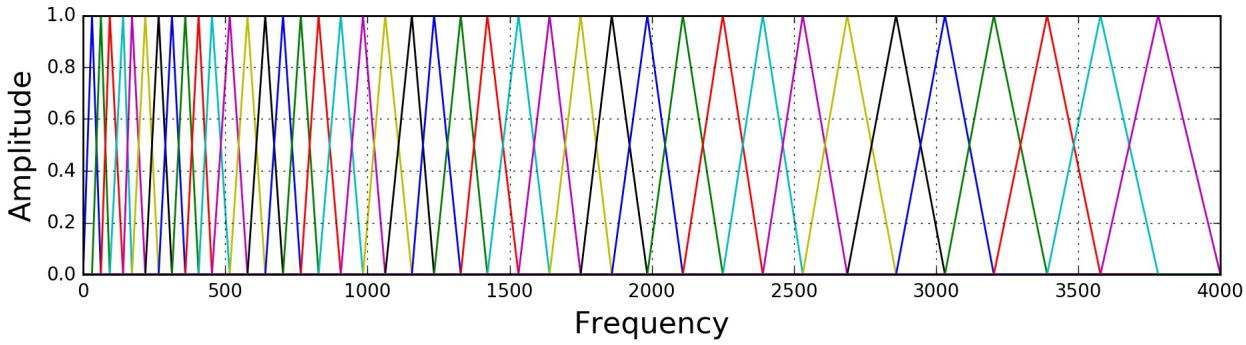

Figure 4.4: Mel-filterbank composed of triangular filters [Fay16]

MFCCs, frequently used features in speech recognition, can be obtained by applying a Discrete Cosine Transform (DCT) to the results gathered after applying mel-filterbanks. DCT is used as a decorrelation method for the mel-filterbanks coefficients. The result of this operation yields the MFCC coefficients as shown in Figure 4.6.

4.4 Deep learning frameworks

As earlier stated in Section 4.1, artificial neural networks aim to mimic humans' neural systems by implementing interconnected artificial neurons. Deep neural networks enhance this concept by increasing the number of neurons in the networks making it closer to its goal of mimicking the human nervous system. Increasing the number of neurons in DNNs also increases the number of parameters that need to be calculated. Calculating a larger number of parameters require additional computing power, which leads to the need to use infrastructures that are suitable for this purpose, both in terms of software and hardware [GBC16].

Central processing units (CPUs) are used to run software on a large range of devices, computers, smartphones, etc, but it has been demonstrated that CPU computing power is insufficient to train ANNs [GBC16]. Graphics processing units (GPUs) are computer components originally designed to execute graphical ap-

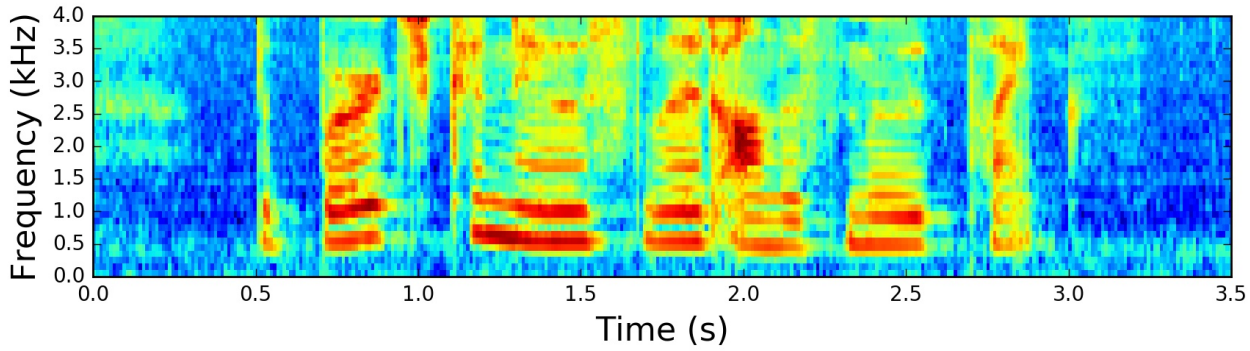

Figure 4.5: Mel-filterbank features [Fay16]

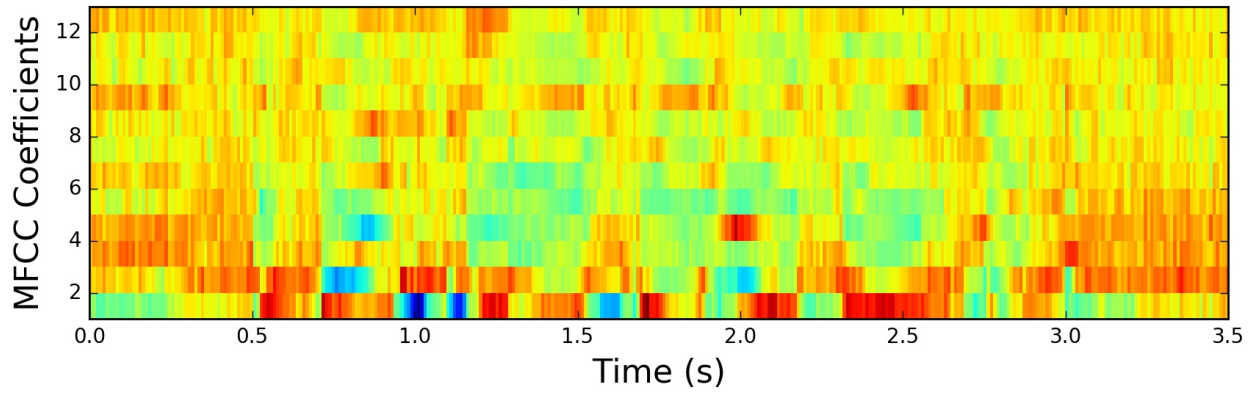

Figure 4.6: MFCC Coefficients [Fay16]

plications, such as video games. Graphical applications may involve matrix multiplications, for example, in video games to convert 3D coordinates into 2D so that they can be displayed on screens. Performing these operations requires large memory buffers and bandwidth and the ability to execute instructions in parallel. To perform such demanding tasks GPUs have memories larger than CPUs' caches, with the downside of having slower clock speeds and less ability to perform branching.

The requirements for neural network algorithms are identical to those just mentioned for graphics algorithms, *i.e.* large memory buffers and parallelism. At each step of the training process, neural networks calculate a large number of parameters, activation function values and gradients of multiple neurons. Large memory bandwidths and parallelism can greatly improve the performance of such operations [GBC16]. Therefore, taking this into account, GPUs have the necessary properties for neural network training.

GPUs were first developed with the special goal of solving graphics tasks. Progressing in time, GPUs started to be employed for more diverse purposes and were no longer just limited to graphics-related operations. The scientific community began employing GPUs to train neural networks in the mid-2000s. Two examples of these employments were a GPU implementation of a two-layer fully connected network, which showed a 3x speedup over the usage of a CPU [SBS05], and the demonstration of GPUs used to accelerate supervised convolutional networks [CPS06].

Shifting from being task-specific components to becoming used in a broader range and more general scenarios rises the necessity of GPUs to be fit and usable for a wider group of users. With this necessity in mind, APIs, frameworks and toolkits have since been developed to ease the development of code that is capable of using the parallelism and large memory buffers provided by General Purpose GPUs (GP-GPUs).

One of the most well-known of these is CUDA, a C-like toolkit and API developed by NVIDIA. Even with tools like CUDA, developing code in an efficient way to run on GPUs is still a difficult task given the differences regarding parallelism and memory between CPUs and GPUs.

Due to the difficulty of developing efficient code for GPUs, it must follow the paradigm of not having to be rewritten in the future, *i.e.* the code should be developed aiming to be reusable. This re-usability can be achieved by compiling commonly used functions and algorithms, such as matrix multiplications, convolution operations, activation functions and gradient methods, etc, in libraries. Following this methodology, code can be developed such that it achieves high performance executing on GPUs and is easy to be used by making calls to library functions [GBC16].

As earlier stated, neural networks greatly improve their performance when trained on GPUs. There is a vast range of machine learning libraries, some of which are targeted to specific types of models, such as neural networks, support vector machines, trees, etc. Regarding the scope of the current, the next paragraphs introduce some of the popular frameworks used in the development of neural networks.

TensorFlow [AAB+15] is an end-to-end open-source platform for machine learning developed by Google that enables beginners and advanced users to develop neural models with a higher or lower level of abstraction. TensorFlow incorporates Keras [Co15] high-level API. With Keras, TensorFlow provides an easy and fast-to-use neural network prototyping and training standard. This standard is used in research projects and in products deployed to production by companies [3]. Another feature of TensorFlow is being easily deployable to a wide range of heterogeneous environments, varying from mobile devices (smartphones and tablets) to highly distributed machines with a large number of components such as GPUs.

PyTorch [PGM+19] is a deep learning framework developed by Meta AI and aims to deliver usability and speed. Based on frameworks like Torch7 [CKF] and TensorFlow [AAB+15], PyTorch provides an array-based programming approach that is GPU-accelerated. PyTorch is developed to fit in the Python ecosystem, *i.e.* to be a Python program. This aims to ease users familiar with the ecosystem, especially researchers, to easily develop their models and integrate PyTorch with other commonly used libraries, *e.g.* data visualisation libraries. Similar to how TensorFlow makes use of Keras, Lightning[4] (previously PyTorch Lightning) integrates with PyTorch to offer a higher-level abstraction. Lightning organises PyTorch code, such as train, validation and test loops, optimisers, schedulers and predictors, into modules in order to eliminate boilerplate and increase scalability[5]. If available, with Lightning GPUs can easily be enabled to accelerate the training process. This can be done by changing the accelerator and strategy arguments passed to the Trainer class.

NVIDIA, a company famously known for developing GPUs, as part of its AI platform, developed NeMo. NeMo is a toolkit/framework targeting deep learning, more precisely the fields of Natural Language Processing, Text-to-Speech and Automatic Speech Recognition. NeMo uses PyTorch and Lightning as its base to offer a collection of modules for each of the research fields just mentioned. By being developed on top of PyTorch and Lightning, NeMo can easily scale model training to be performed not only on multi-GPU systems but also on multi-node clusters, *i.e.* multiple high-performance machines interconnected. NeMo also uses Hydra [Yad19], a framework developed by Facebook Research. This framework allows complex models, such as NeMo's, to be configured using command line arguments or configuration files. The components needed to set up a model architecture, such as the encoder and decoder, the optimiser, etc, can be specified in a configuration file. This feature helps to reduce, even more, the boilerplate code needed to develop a deep neural model. An example of a configuration file used to build a NeMo model using the QuartzNet15x5 architecture can be seen in Appendix A.3.

[3] `https://www.tensorflow.org/overview`
[4] `https://lightning.ai/`
[5] `https://pytorch-lightning.readthedocs.io/en/stable/starter/introduction.html`

4.5 Data-centric

Machine learning can be described as the process in which an algorithm is applied to a dataset in order to produce a model of such data. The models generated by this process can be tuned in order to improve their performance, *i.e.* to improve how well the models represent the given data.

The tuning process is usually focused on the algorithm. Accordingly to a chosen criteria, *e.g.* the problem context and respective goal, some algorithm parameters are chosen in order to find the optimal combinations of their values. The search for the best combination is usually done using one of two methods:

- Grid Search - iteration over a set of value combinations of the selected parameters
- Random Search - randomly assigning values to the selected parameters in order to try to avoid local maximums/minimums in which the grid search approach may fall into

Figure 4.7 presents a pipeline that describes a methodology based on algorithm tuning.

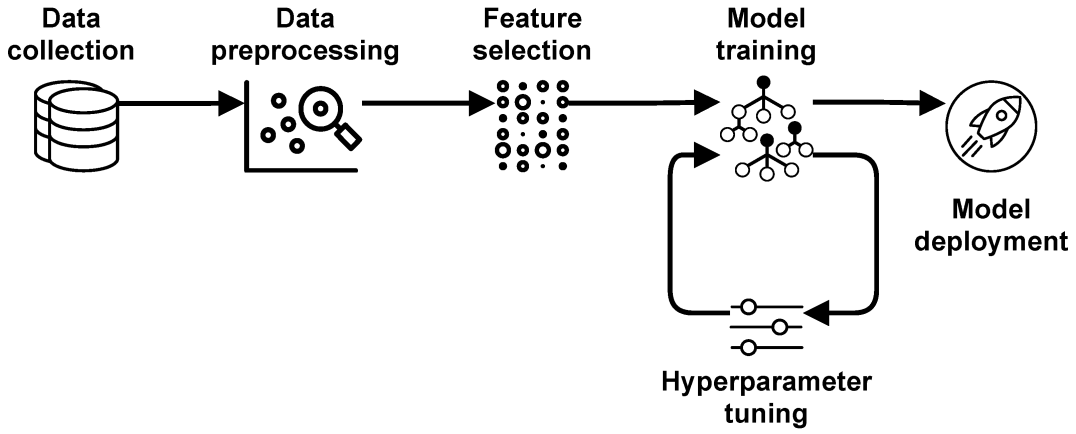

Figure 4.7: Pipeline of methodology based on algorithm tuning [GTK22]

As just stated, models are generated by applying algorithms to datasets, therefore, another possible strategy to improve the results yielded by models is by focusing on a data-centric approach, *i.e.* focusing on the data instead of the algorithm.

A data-centric approach aims to transform data so that the quality of the final dataset is higher than the original. This raises the question of how data quality can be defined and measured. Data quality can be described regarding two separate topics, the structure of a dataset, and the characteristics of the data.

Being data a crucial component for creating machine learning models, datasets must have sufficient entries to enable algorithms to provide an accurate representation of the data. Larger isn't always better [WŠW+21], datasets with vast amounts of data must be well-organised and have an easy-to-work structure in order to facilitate the analysis and manipulation of its contents. Despite the scarcity of standards, data can be collected and stored using different guidelines specific to different companies and/or projects. Not following any standard, although well-organised some structures can contain directories or file names which are not clear. Therefore, in addition to being well-organised and structured datasets should also be well-documented.

In addition to how the data is structured, organised and documented, the data contents can also be classified as good or poor regarding its quality. Obtaining data is a very delicate process since errors could

lead to noise being introduced in the dataset.

Data noise, *i.e.* invalid data that overlaps valid one, can be present in various forms, from data corruption, *e.g.* bits get corrupted when transmitting data through a network, to missing values, incorrect information such as typing errors, and duplicate entries. Outliers are data points that differ significantly from the majority of the data. These entries can either occur naturally or by data corruption. In the first case characteristics of individual data entries just don't fall in line with the majority and therefore are distant. In the second one, the values of the attributes might have been modified and thus setting these points further away from the majority.

Data entries with missing values can appear either by data corruption or because some values were not collected. Another example can be missing labels due to human error. This problem can be resolved using methods such as estimating the missing values and ignoring or removing the respective entries.

Creating a dataset might involve obtaining data from various sources which can lead to duplicate entries, *e.g.* a person can be inquired twice. Similar data entries should be evaluated to determine their similarity value, *i.e.* to measure how distant they are. When the distance between both entries reaches a threshold, it can be determined to either merge or keep the entries as is.

When a dataset contains a large number of missing values, duplicate entries, or noise, such as data corruption and outliers, the data content might become ambiguous. Having ambiguous data makes it unclear if the dataset is fit to solve a specific problem.

Another aspect related to the quality of a dataset is the fairness and representativeness of its content. Considering a dataset well-organised, structured and documented and with levels of noise and outliers within a threshold. The dataset's quality could still be considered "poor" if the data doesn't have the desired representativity, *i.e.* the balance between the data samples is not favourable to solve the problem being addressed. Such a problem can be addressed by either discarding the dataset or by focusing on collecting specific data in order to achieve the desired representativity [WŠW+21].

Validating data is an important and delicate task. When not properly validated data might be used incorrectly. Unvalidated data can either not be suitable to solve the problem or have undesired characteristics, such as large amounts of noise. Data augmentation methods or generative models generated new data based on existing data. Such systems generate new data instances according to the ones provided to them. Therefore, when the original data is not validated, the generated data will contain undesired characteristics present in the original, such as noise. This process is very delicate since increases the amount of data but also amplifies its characteristics. When data is not carefully validated, the data augmentation processes can reduce the overall quality of a dataset by augmenting characteristics such as noise, which in turn will affect the performance when creating/tuning models [BZP+19].

In summary, data quality can be defined on top of the previously mentioned characteristics

- Organisation, structure and documentation
- Data noise and outliers
- Data fairness and representativity

which have the flexibility of being more or less relevant depending on the problem and the respective goal and the approach to be taken.

In [GTK22] Garan *et al.* present the current view of the scientific community on the data-centric methodology and propose a new implementation. Figure 4.8 illustrates a pipeline of the community's current view

of the data-centric methodology. This pipeline consists of performing a complete machine learning cycle for each data-oriented scenario. Such an approach is both computationally demanding and time-consuming since algorithm tuning takes place for every data preprocessing/feature selection scenario.

To overcome these issues, Garan *et al.* proposed a new pipeline illustrated in Figure 4.9. The proposed implementation tackles the computational power and time issues by performing algorithm tuning just once. A base model is defined in the initial stage of the pipeline accordingly to the final model to be used in order to evaluate each of the data-driven scenarios. A final model is then trained and tuned using the data-oriented scenarios that yielded the best results.

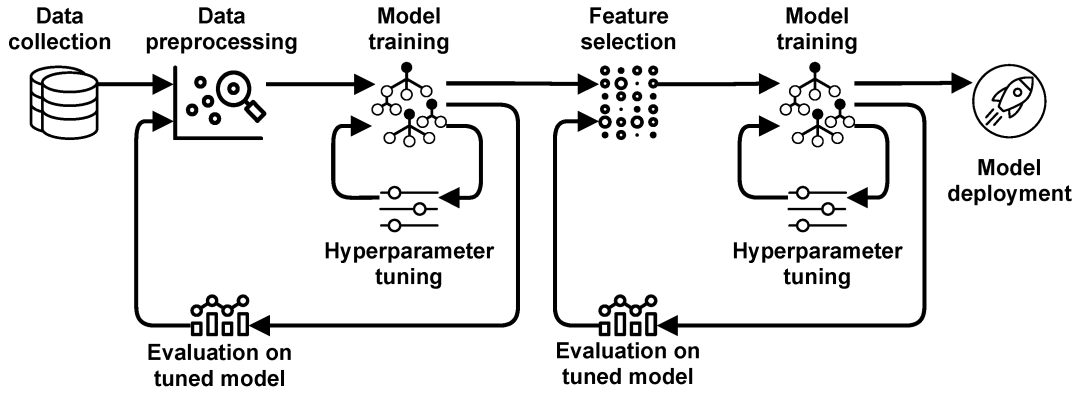

Figure 4.8: Data-centric pipeline with complete machine learning cycle [GTK22]

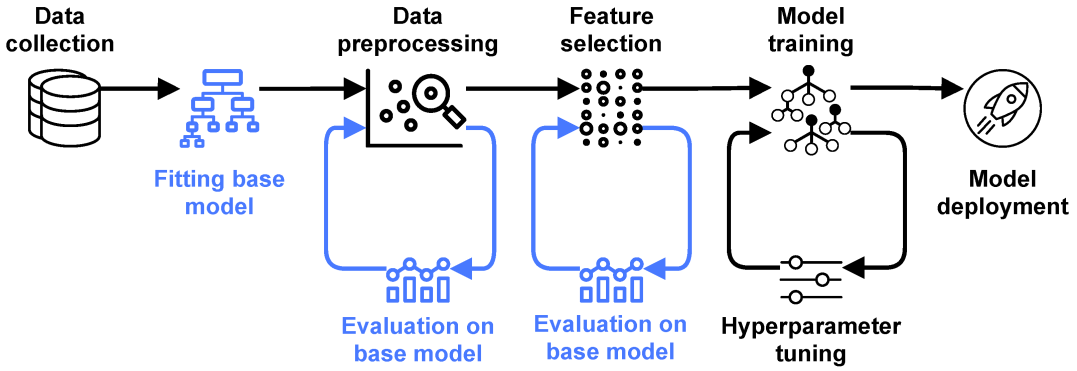

Figure 4.9: Data-centric pipeline using a base model [GTK22]

5

Proposed System

As previously demonstrated, specific frameworks and approaches are utilised to create deep learning models. The current Chapter describes the proposed system of the current work in:

- 5.1 - **System objectives** - description of the goals to be addressed by the system
- 5.2 - **Architecture** - characterisation of the architecture adopted for the ASR system
- 5.3 - **Design** - description of the framework and architecture used for the developed models
- 5.4 - **Overview** - final overview of the system

5.1 System objectives

The current work aims to create an Automatic Speech Recognition deep neural model that transcribes European Portuguese speech into the respective transcription. Creating a final model which achieves such goal implies several stages. These stages represent a variety of modules, from dataset acquisition and data

pre-processing to model training and evaluation. In the development of this work, in order to connect these stages to achieve the final goal, the stages were assembled into a pipeline, described in Section 5.2. Section 5.3 describes the framework and architecture used to develop the deep neural models.

5.2 Architecture

To achieve the goal of creating models capable of transcribing Portuguese audio into the respective transcription using deep learning, a pipeline was set up to facilitate connections between each necessary stage of this process. Figure 5.1 illustrates the pipeline built for the project whose stages are described below:

- **Dataset Acquisition** - this stage aims to explore and acquire new speech datasets in Portuguese (audio recording files and respective transcriptions)
- **Content Analysis** - dataset analysis, assessment of the dataset's initial structure, documentation, content quantity and quality and audio file encodings
- **Pre-Processing** - dataset restructuring, data cleaning and manifest creation
- **Model Creation** - creation of models with pre-processed data
- **Results Evaluation** - testing and evaluation of the created model

Dataset Acquisition and Content Analysis

After obtaining a dataset candidate to fulfil the system's objective, its contents are analysed to ensure they satisfy quality and quantity requirements for model creation. In this stage, it is evaluated the number of hours of audio recordings, the speech noise of small batches of data samples and the quality of the transcriptions regarding its annotations.

Pre-Processing

If the candidate dataset fulfilled the previous verifications, the following step is to modify the dataset as needed. These modifications can be restructuring the dataset in order to better integrate with further data processing scripts and to clean data noise or change some of its characteristics. Examples of data noise-cleaning are removing odd characters in the transcriptions or selecting only a set of pretended data samples. Examples of data modification changing audio encodings and sampling rate.

Model Creation

After the data is properly organised and adjusted, the next stage is to train a model using the chosen framework and architecture.

Results Evaluation

As a final, the model is tested and evaluated. The pipeline restarts according to the results yielded, *e.g.* the results might indicate that a new analysis is required or other pre-processing of the data should be applied.

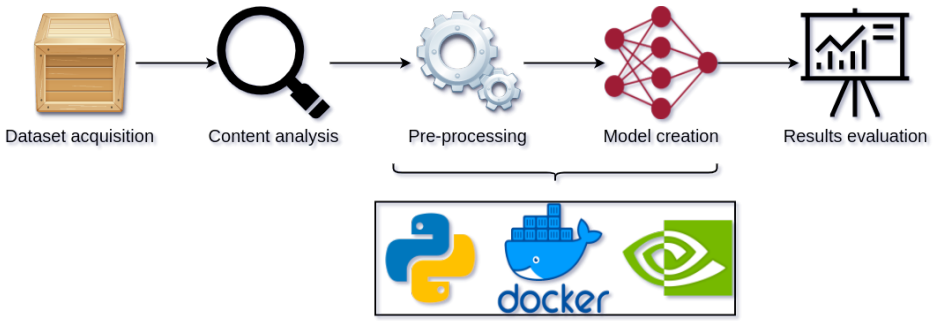

Figure 5.1: Work pipeline and toolkit

5.3 Design

As examined in Section 4.4 a variety of frameworks is available to develop deep neural models. Section 5.3.1 describes the framework used for model creation and evaluation and Section 5.3.2 describes the architecture of the models developed in the present work.

5.3.1 NVIDIA NeMo

State-of-the-art deep learning frameworks explore hardware features to their full potential. Multi-CPU, multi-GPU, and multi-node with high-speed interconnectors allow algorithms to perform faster calculations and therefore bring down the training time of models.

NVIDIA **NeMo**[1] [KLN+19] is framework developed by NVIDIA built on top of the **PyTorch** and the **PyTorch Lightning** frameworks (Figure 5.2) and is meant "*(...) for building, training, and fine-tuning **GPU-accelerated** speech and natural language understanding (NLU) models with a simple **Python** interface.*"[2]. NeMo provides separate collections for Automatic Speech Recognition, Natural Language Processing, and Text-to-Speech models. Each collection consists of prebuilt modules that include everything needed to train new models. Every module can easily be customised, extended, and composed to create new conversational AI model architectures[3].

Figure 5.2: NeMo Integration with PyTorch and PyTorch Lightning

Given the infrastructure in which the present was being developed, shown in Section 7.1, and the features made available by NeMo, NeMo was the framework selected to develop the current work models.

[1] https://github.com/NVIDIA/NeMo
[2] https://developer.nvidia.com/nvidia-nemo
[3] https://docs.nvidia.com/deeplearning/nemo/user-guide/docs/en/stable/starthere/intro.html

5.3.2 Model architecture

Regarding NeMo's automatic speech recognition collection, Jasper [LLG+19] and QuartzNet [KBG+19] are two end-to-end model architectures worth mentioning due to their role in the current work.

Jasper consists of a block architecture designed to ease fast GPU inference. The block architecture is composed of one convolutional pre-processing block, B×R blocks (where B represents the number of blocks and R the number of sub-blocks) and three convolutional post-processing blocks. Each block input is connected to the last sub-block using a residual connection. Each of the R sub-blocks applies a set of four operations: a 1D convolution, batch normalisation, ReLU and dropout. Models with this architecture are trained using the CTC loss function. Jasper models achieve state-of-the-art results of 2.95% when using a beam-search decoder and an external language model.

The QuartzNet architecture has the goal of achieving state-of-the-art results while using smaller models. Smaller models have fewer parameters, therefore, they are easier to train and allow deployment on less powerful machines. QuartzNet architecture is based on Jasper's B×R block architecture. It also consists of one convolutional pre-processing block, B×R blocks, and three convolutional post-processing blocks. Like Jaspers, QuartzNet is also trained using the CTC loss function. The R sub-blocks perform similar operations the only difference being the replacement of the 1D convolution with a 1D time-channel separable convolution (comparison illustrated in Figure 5.3). The 1D time-channel separable convolution is an implementation of depthwise separable convolutions which can be separated into two other convolutions, a 1D depthwise convolution and a pointwise convolution (see Figure 5.4). The 1D depthwise convolutional layer performs convolutions across time, and a pointwise convolutional layer performs a 1×1 convolution across features/channels [KBG+19, MVKK]. Changing the type of convolution used in the sub-blocks enables the QuartzNet architecture to achieve state-of-the-art results and drastically reduce the number of parameters. Table 5.1 shows a comparison between the results and the number of parameters of the Jasper and QuartzNet architectures.

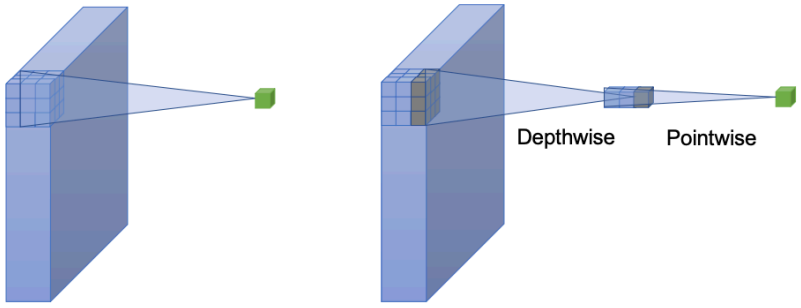

Figure 5.3: Standard convolution (left) and depthwise separable convolution (right) [GLF+]

	Parameters	WER
Jasper	333 millions	2.84%
QuartzNet	19 millions	2.69%

Table 5.1: Performance (WER) of QuartzNet and Jasper architectures on the LibriSpeech dataset (Table 4 from [KBG+19])

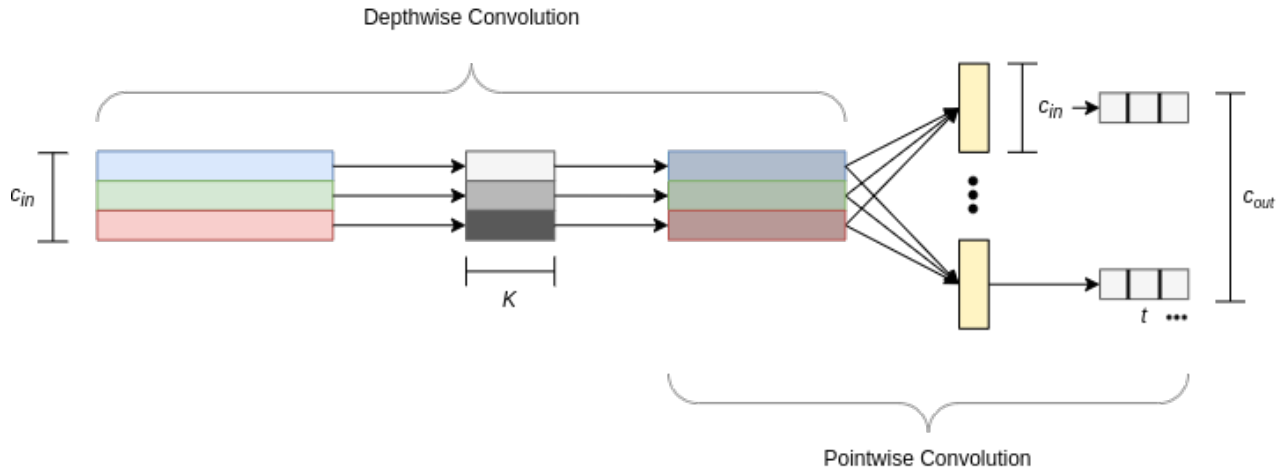

Figure 5.4: 1D time-channel separable convolution

5.4 Overview

The proposed system is capable of transcribing Portuguese audio into the respective transcription while using deep neural models with a reduced number of parameters. This is achieved through the use of a pipeline responsible for connecting each module of the system, and due to the use of a framework that enables the development of models with the chosen architecture.

6

System Implementation

To develop deep neural models the system previously proposed in Chapter 5 was implemented. This Chapter outlines the methodology used to accomplish such implementation. This is described in:

- 6.1 - **Docker environment** - the Docker environment where the system will be developed and executed
- 6.2 - **Developed software** - description of the two types of software developed
- 6.3 - **Data pre-processing** - description of changes made to data with pre-processing software
- 6.4 - **API and web interface** - description of the developed API and web interface
- 6.5 - **Implementation issues** - review of the problems encountered during the system's implementation

6.1 Docker environment

Being able to replicate the pipeline environment is crucial so experiments can be replicated in different machines. Docker allows this by running applications inside containers, *i.e.* a self-contained environment, isolated from the host operative system and other containers. A Dockerfile (Appendix A.5) was created in order to set up the experiments' environment based on NVIDIA's NeMo framework. The Dockerfile contains a set of instructions for Docker to be able to build the image of the environment in which the project's pipeline stages will be executed.

The image used to execute the pipeline stages of the project, namely the pre-processing, model creation and model evaluation stages, uses the PyTorch image[1] (NeMo's base framework) as starting point. On top of PyTorch's image, required libraries and updates are installed in order for NeMo to be used in the most efficient making the best use of GPUs. The list of relevant modifications can be seen below.

- Installation of NeMo requirements[2]:
  - libsndfile1
  - Cython (a superset of the Python language that additionally supports calling C functions and declaring C types on variables and class attributes[3])
- Installation of NeMo
- Updates of the pre-installed PyTorch libraries

6.2 Developed software

The work developed in the first two stages of the developed pipeline, *i.e.* Dataset Acquisition and Content Analysis (Figure 5.1), is mostly done without resorting to any software. However, the Pre-processing, Model Creation and Results Evaluation stages required software to be developed. The software developed for these three stages is presented and divided into two groups:

- **Data pre-processing software** - software developed for the Pre-processing stage
- **Model software** - software developed to create and evaluate models in the Model Creation and Results Evaluation stages respectively

6.2.1 Data pre-processing software

Under this category fall two types of software: the software developed to restructure the datasets, and the software developed to generate the manifest files used as input for the model creation software.

Both types of software were developed separately for each of the datasets whose analysis from the previous stage was positive regarding their use in the current work. The information retrieved during the analysis stage is used to determine if the dataset needs restructuring and which data samples will be selected to be

[1] `https://catalog.ngc.nvidia.com/orgs/nvidia/containers/pytorch`
[2] ffmpeg was not installed due to incompatibility
[3] `https://cython.org/`

used in the stage of model creation. Datasets lack structure standardisation, therefore, for each dataset, Python scripts were developed separately in order to perform dataset restructuring and data selection.

To load the input data the NeMo framework uses manifest files. These files contain one entry per audio file which contains the audio file path, audio duration, offset and audio transcription. Listing 6.1 shows an example of a manifest file entry.

```
{
    "audio_filepath": ".../a10041a1.wav",
    "duration": 1.28,
    "offset": 0,
    "text": "repetir"
}
```

Listing 6.1: Example of manifest entry

6.2.2 Model software

This category includes software developed for training and evaluating models. For both training and testing scenarios, Python scripts were developed using the NeMo framework.

Appendix A.1 shows the code developed for model training. This script is used in the training-from-scratch and transfer learning experiments. By using Hydra, a framework "*(...) that simplifies configuration for complex applications*"[4], NeMo is able to use the same code for both training from scratch and transfer learning purposes. Using external configurations eases the development and setup of neural models, especially with complex architectures such as QuartzNet. The configurations used by NeMo specify the components required by the PyTorch library to train deep neural models. The list below presents the fields and respective descriptions used in the configurations for the current work:

- `model` - specifies the encoder, decoder, output labels, batch size, data pre-processors and augmenters, optimisers and paths to the train, validation and test subsets
- `trainer` - sets the parameters for the trainer, being the more relevant the number of epochs, the number and the parallelisation strategy
- `init_from_nemo_model` - when present, the code performs transfer learning using the specified model as a starting point, else performs train from scratch
- `exp_manager` - experiment logging configuration
- `hydra` - provides configurations for the Hydra framework

In the current work, the default configuration for the QuartzNet15x5 architecture was used. This configuration can be found in NeMo's GitHub repository and is also exemplified in the Listing of the Appendix A.3. Some relevant fields of the configuration used are:

- Data pre-processor - **AudioToMelSpectrogramPreprocessor** - converts wavs to mel spectrograms

[4] `https://docs.nvidia.com/deeplearning/nemo/user-guide/docs/en/stable/core/core.html#configuration`

- Data augmenter - **SpectrogramAugmentation** - using Cutout [DT17] randomly masks out regions of input during training
- Optimiser - **NovoGrad [GCH+19]** - stochastic gradient descent method
- Scheduler - **Cosine Annealing [LH]** - An learning rate starts high and is then quickly reduced to a minimum and then quickly boosted again which provides a "warm restart"

QuartzNet architecture (Figure 6.1) is based on Jasper's B×R block architecture. QuartzNet consists of one convolutional pre-processing block, B×R blocks, and three convolutional post-processing blocks trained with the CTC loss function. A residual connection connects each block input to the preceding sub-block and each of the R sub-blocks performs the following four operations:

1. K-sized depthwise convolutional layer with c_{out} channels
2. pointwise convolution
3. normalization
4. ReLU

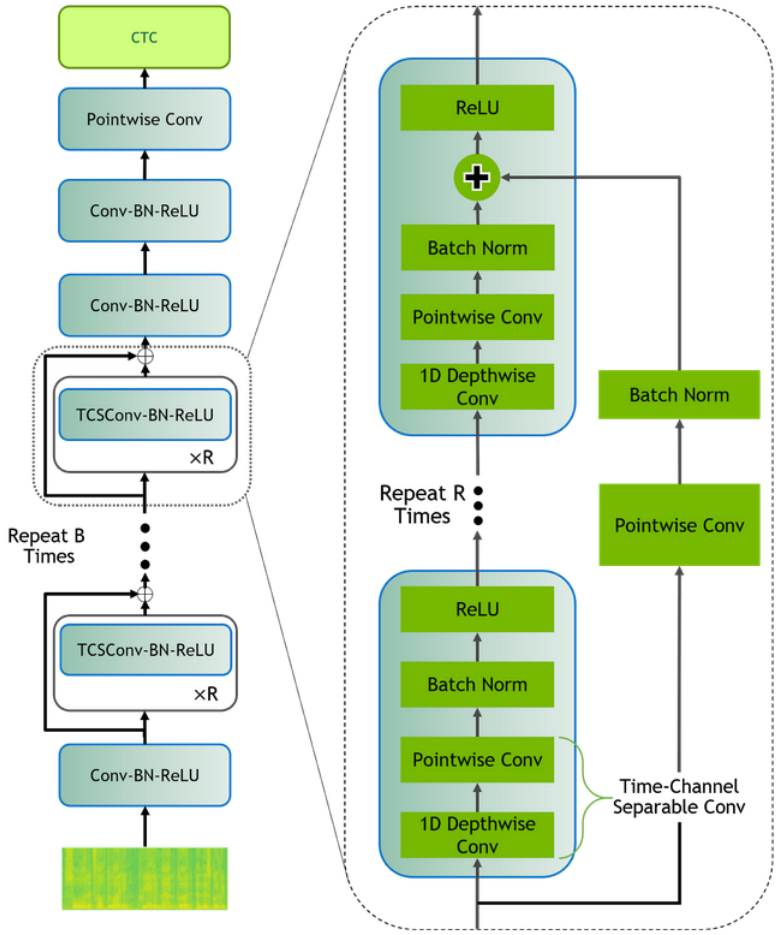

Figure 6.1: QuarzNet BxR architecture [KBG+19]

The 1D depthwise convolutional layer performs convolutions across time, and a pointwise convolutional layer performs a 1×1 convolution across features/channels [KBG+19, MVKK]. As already stated, the final part of the architecture consists of three additional convolutions. Table 6.1 presents the repetitions (of block and sub-blocks), kernel sizes, output channels and the total number of parameters of three different variants of the QuartzNet architecture. As mentioned, the current made use of the variant QuartzNet15x5.

Block	**R**	**K**	**C**	**S**		
				5x5	**10x5**	**15x5**
C_1	1	33	256	1	1	1
B_1	5	33	256	1	2	3
B_2	5	39	256	1	2	3
B_3	5	51	512	1	2	3
B_4	5	63	512	1	2	3
B_5	5	75	512	1	2	3
C_2	1	87	512	1	1	1
C_3	1	1	1024	1	1	1
C_4	1	1	||labels||	1	1	1
Params, M				6.7	12.8	18.9

Table 6.1: QuartzNet Architecture. The model starts with a conv layer C_1 followed by a sequence of 5 groups of blocks. Blocks in the group are identical, each block B_k consists of **R** time-channel separable **K**-sized convolutional modules with **C** output channels. Each block is repeated **S** times. The model has 3 additional conv layers (C_2,C_3,C_4) at the end. [KBG+19]

Appendix A.2 presents the code developed for evaluating the trained models. This code evaluates the selected model using the WER and ROUGE metrics. This script can additionally be used to transcribe single audio files specified through execution parameters.

For easier and more intuitive use the software described in this Section is accompanied by a `Makefile` (Appedix A.4). The **`make`** command should be executed followed by the target of the desired operation, **`train`**, **`transfer`**, **`test`** or **`transcribe`**. In turn, the operation target must be given the required parameters, which are: `CONFIG_PATH` for both the `train` and `transfer` targets; `TEST_PATH` and `MODEL_NAME` for `test`; `MODEL_NAME` and `AUDIO_PATH` for `transcribe`.

6.3 Data pre-processing

Datasets often lack structure standardisation, therefore there is usually the need to reorganise them to improve workflow and compatibility with data processors. This section presents changes made to the structure and data of the SpeechDat dataset.

6.3.1 SpeechDat original directory structure and data

The Portuguese database from the SpeechDat European Project was developed between March 1st 1996 and February 28th 1998. This database represents the SpeechDat dataset in the current work. The original structure was composed of 11 different zip files. Each zip file contained one ISO file (CD image). Among the ISO files were distributed 49 blocks (sub-directories). Each block was divided into session sub-directories which in turn contained the respective audio recording files, as can be seen in the Listing 6.2. The original

audio recordings were encoded in A-LAW [5], an algorithm used for encoding audio signals, in particular, voice encoding, with a sampling rate of 8 kHz 8-bit. In addition to containing a partition of the 49 blocks, each ISO file also contained a copy o the following four sub-directories:

- **doc/** - directory with documentation regarding the construction and structure of the dataset
- **index/** - directory with content file, *i.e.* file indexing the dataset with the path, transcript and other information regarding each audio recording
- **source/** - empty directory
- **table/** - directory of files regarding the sessions, speakers and lexicons information

```
SpeechDatII/
     FIXED1PT_01.7z
     ...
     FIXED1PT_11.7z
         FIXED1PT_11.ISO
             fixed1pt/
                 copyrigh.txt
                 disk.id
                 readme.txt
                 block46/
                 ...
                 block48/
                        ses4800/
                        ...
                        ses4841/
                            a14841w4.pta
                            a14841w4.pto
                            ...
                 doc/
                 table/
                 source/
                 index/
                        ...
                        contents.lst
```

Listing 6.2: Original SpeechDat directory tree

6.3.2 SpeechDat modified directory structure and data

Structure

The original structure of the dataset was complex and challenging to work with, therefore there was a need for restructuring to ease the use of the dataset with the modules of the proposed pipeline. The contents of the four folders previously mentioned were identical among all ISO files, thus, these were moved to the root

[5] https://en.wikipedia.org/wiki/A-law_algorithm

directory of the dataset "SpeechDatII/" (except "source/" since it was empty in all ISO files). Following up, the respective sessions and audio recordings of each of the 49 blocks, which were in different ISO files, were moved into a single sub-folder, "FIXED1PT/", in the root directory. The resulting dataset structure can be seen in Listing 6.3.

```
SpeechDatII/
    FIXED1PT/
        block00/
        ...
        block48/
            ses4800/
            ...
            ses4841/
                a14841a1.pta
                a14841a1.pto
                a14841a1.wav
                ...
    doc/
    table/
    index/
        ...
        contents.lst
        contents.csv
        contents_quality.csv
```

Listing 6.3: New SpeechDat directory tree

Data

After restructuring, the `contents.lst` file present in the **`index/`** sub-folder was also subject to changes. Its encoding was changed from `ISO-8859-1` to `UTF-8`. The values of some fields were subject to minor adjustments, *e.g.*, in the field `path`, the backslashes (\) were replaced with (/) so they were compatible with the filesystem of DGX OS 5.2.0, based on Ubuntu 20.04.4 LTS[6]. The updated content of the `contents.lst` files was stored in the `contents.csv` file.

For each audio recording, the `contents.csv` file contains its path in the dataset, a corpus code (special label for the type of recording), the region and gender of the speaker and the transcription of the audio. Some transcriptions were presented with special tokens (presented below). These tokens described word mispronunciation, truncation or unintelligible stretches of speech, speaker or background noise, or pauses.

- Tokens describing errors in pronouncing words:
  - Mispronunciation (*)
  - Unintelligible stretches of speech (**)
  - Truncation (~)
- Tokens describing noise:

[6] `https://docs.nvidia.com/dgx/dgx-os-release-notes/index.html#release-5`

  - Stationary noise (`[sta]`)
  - Speaker noise (`[spk]`)
  - Intermittent noise (`[int]`)
  - Filled pause (`[fil]`)

Besides the original audio recordings, each audio was also accompanied by a SAM label file, "`.pto`", which contained the following sets of rows with information regarding the audio recording:

- identification
- session
- recording conditions
- speaker
- file

The information from the file rows contained an assessment code which had one of the values described below. This value was used to label each audio signal regarding its recording quality. For each audio recording present in the `contents.csv` file, an additional column was added with this labelling. The updated content is stored in the `contents_quality.csv` file under the `index/` folder.

- **OK** - includes clean audio recordings ready to be used
- **NOISE** - includes audio recordings with some speaker or background noise
- **GARBAGE** - includes empty audio recordings, audio recordings with no transcription and audio recordings with a large number of errors pronouncing words or unintelligible speech
- **OTHER** - includes audio recordings with unknown problems, with it being only noise, some empty transcriptions and others

After the analysis stage, it was concluded that some of the data wouldn't be fit for the current work. The list below shows the characteristics of the audio recordings which were left out, *i.e.* were never used in this work experiments. All the audio recordings which didn't include any of these characteristics were marked as valid. This information was stored in the `contents_quality.csv` file under the `index/` folder.

- audio recordings whose quality label was GARBAGE or OTHER
- audio recordings whose transcriptions was ** or empty
- audio recordings whose transcriptions contained ~ or *

The original audio recordings were encoded in A-LAW with a sampling rate of 8 kHz 8-bit. In this pre-processing stage, the audio recordings marked as valid in the `contents_quality.csv` file were re-encoded in the WAV encoding with a sampling rate of 16 kHz. The audio recordings were re-encoded and re-sampled in order to match the formats accepted by the framework and the sampling rate of pre-trained models.

After this re-encoding and re-sampling, each audio recording was added to a manifest file as shown in Listing 6.1.

Two separate "versions" of the dataset were used in this work. The first version consisted of the dataset with the restructuring and data pre-processing just mentioned. The second version had an additional pre-processing in which the previously mentioned noise tokens, `[sta]`, `[spk]`, `[fil]` and `[int]`, were removed from the transcriptions.

6.4 API and web interface

An API and a simple web interface were developed to allow a user-friendly experience for audio transcription. The API was run using the same Docker image used for training and testing.

The `run.sh` script setups and starts the API by performing the following steps:

- Installation of the requirements, `FastAPI` and `Python-Multipart`
- Fetch of the environment variables in the `.env` file, which are the host's IP address and port
- Launches the API which can be accessed at `localhost:28080`.

The API provides one `POST` endpoint, `/transcribe/`. This endpoint receives a file as the value of the field `audio_file_path` which should be encoded as `multipart/form-data` and returns the transcription in JSON format, as shown in Listing 6.4.

```
{
    "transcription": "exemplo de audio"
}
```

Listing 6.4: API transcription example

A demo of this API is live at http://voice.xdi.uevora.pt and is shown in Figure 6.2.

6.5 Implementation issues

Throughout the implementation of the system, some issues rose regarding the data and its encoding and the infrastructure where the system was run.

One of the main issues of this implementation was the encoding of the contents files, originally being `ISO-8859-1`. Some research had to be done to ensure a proper conversion from `ISO-8859-1` to `UTF-8` to avoid any data loss.

The resources of the infrastructure where the system was built and tested are managed by the Slurm Workload Manager[7] [JG03]. Due to the SLURM configuration in this infrastructure being in an early stage, and the experience with SLURM being very basic, some issues rose when configuring SLURM batch scripts to use the system. Examples of such difficulties were the use of multiple resources, such as CPUs, GPUs and especially multiple computational nodes, to either train or test the models.

[7] `https://slurm.schedmd.com/`

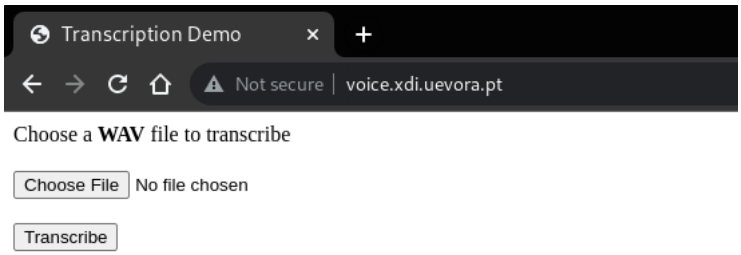

Figure 6.2: ASR API Demo - Web Interface

7

Experiments

Training deep learning models is very computationally demanding. To speed up this task, GPUs are a good replacement for CPUs due to being more fit to execute parallel tasks and having a higher memory bandwidth. The high level of parallelism provided by these devices is ideal for training machine learning models as they can process multiple calculations simultaneously. Additionally, the GPU's large memory bandwidth is appropriate for working with the large amounts of data commonly used in this task.

This Chapter describes:

- 7.1 - **Infrastructure** - the infrastructure used in the experiments developed in the current work
- 7.2 - **Experiments** - presentation of the different types and sets of experiments and respective results and analysis
  - 7.2.1 - **Metrics** - definition of the metrics used to evaluate experiments
  - 7.2.2 - **Train from scratch** - experiments based on when the whole model was trained from the beginning
  - 7.2.3 - **Transfer learning** - experiments based on model creation using the transfer learning technique

7.1 Infrastructure

The pipeline (Figure 5.1) described in Section 5.3 was run in the Vision Supercomputer[1]. Vision is an HPC cluster made of 2 compute nodes, interconnected with 8 x 200Gb/s HDR InfiniBand links for parallel processing. The compute nodes are NVIDIA DGX A100 systems, with the following characteristics:

- GPUS: 8x NVIDIA A100 40GB Tensor Core GPUs
- GPU Memory: 320GB total
- CPU: Dual AMD Rome 7742, 128 cores total
- Networking (clustering): 8x Single-Port NVIDIA ConnectX-6 VPI 200Gb/s InfiniBand
- Networking (storage): 1x Dual-Port NVIDIA ConnectX-6 VPI 200Gb/s InfiniBand

Data storage in Vision is shared across all nodes, using NFS over a 200Gb/s HDR InfiniBand link. To minimise any I/O bottleneck when transferring data from the storage to the GPUs, the compute nodes use NFS caching to mount the shared storage. The cluster resources are managed by the Slurm Workload Manager[2] [JG03], which is responsible for allocating the resources requested by each user for each job.

The results presented in this work were obtained in a single compute node: 8 x NVIDIA A100 GPUs, 256 CPUs and 1TB of RAM).

7.2 Experiments

In machine learning, creating models can be done following different paradigms. Two examples of these are, from scratch and transfer learning. Following the from-scratch paradigm, an algorithm is fed with data and the model is generated without any prior work done to it. In transfer learning, a new model is developed using a previously developed model as a starting point. The base model is then tuned during training accordingly to new data which is fed to the algorithm. Both of the previous paradigms were used in the experiments developed in the present work.

Data is a key element in the development of an ASR system based on deep learning, however, there is not much speech data available for Portuguese. Usually, large datasets in Portuguese are private, which means their use is restricted (to certain groups or by payment of service). In the experiments developed in the present work models were developed using the two datasets previously mentioned in Chapter 3:

- SpeechDat dataset ($\simeq$186 hours of audio recordings)
- Multilingual LibriSpeech [3] ($\simeq$168 hours of audio recordings)

The version of Python, Docker image, framework and libraries used in the experiments developed in Sections 7.2.2 and 7.2.3 were the following:

- `Python` - 3.8.10

[1] `https://vision.uevora.pt/`
[2] `https://slurm.schedmd.com/`
[3] `https://www.openslr.org/94/`

- **PyTorch Docker image** - 21.08-py3
- **NeMo** - 1.7.2
- **libsndfile1** - 1.0.28-7
- **Cython** - 0.29.24
- **torch** - 1.11.0+cu113
- **torchvision** - 0.12.0+cu113
- **torchaudio** - 0.11.0+cu113

7.2.1 Metrics

Speech recognition models' performance is evaluated according to the transcriptions generated by the models on given test sets. Metrics are used to evaluate performance by measuring the distance between the transcription generated by the model and the real one. The measurement of this distance differs accordingly to the metric used. In the present work two metrics were used to evaluate the developed models:

- Word Error Rate (WER)
- Recall-Oriented Understudy for Gisting Evaluation (ROUGE)

WER is a metric for evaluating speech-to-text models and is defined as shown in Equation 7.1. This metric determines the distance between transcripts by evaluating substitutions, insertions, and deletions made to a single word or word sequence in order for the transcripts to match.

$$WER = \frac{Substitutions + Insertions + Deletions}{Number\ Of\ Spoken\ Words} \tag{7.1}$$

WER results range from 0 to 1 (or 0% to 100%), but these results can be higher than the upper bound if the number of additional insertions, substitutions or deletions required for the transcripts to match is very large.

ROUGE consists of a package of several methods for the evaluation of summaries. The method used in the present work was ROUGE-L which compares the longest common sequence (LCS) between two summaries/transcripts [Lin04].

Given two summaries X (reference) and Y (prediction) we have:

- Equation 7.2 - represents the recall - evaluates the model's performance capturing all of the information contained in the reference
- Equation 7.3 - represents the precision - allows evaluating the model regarding the generation of more information than the contained in the reference, *i.e.* extra words
- Equation 7.4 - represents the F-measure and the definition of ROUGE-L - used to evaluate the model regarding capturing as many words as possible (recall) but without outputting irrelevant words (precision) [Lin04, Bri21]

$$R_{lcs} = \frac{LCS(X,Y)}{length(X)} \tag{7.2}$$

$$P_{lcs} = \frac{LCS(X,Y)}{length(Y)} \tag{7.3}$$

$$F_{lcs} = \frac{(1+b^2)R_{lcs}P_{lcs}}{R_{lcs} + b^2 P_{lcs}} \tag{7.4}$$

7.2.2 Train from scratch

The MLS dataset contains an unbalanced mix of European ($\simeq$1 hour of audio recordings) and Brazilian Portuguese ($\simeq$167 hours of audio recordings).

The Portuguese variants were divided into different subsets of the MLS dataset for these experiments. Table 7.1 shows these experiments where the European Portuguese subset is represented by "PT" and the Brazilian Portuguese subset by "BR". Both variants of Portuguese were used separately and together in order to carry out different sets of experiments and find out how data quantity affects the performance of deep models developed from scratch.

Experiment	Train	Validation	Test
MLS_S1	PT	PT	PT
MLS_S2	BR	BR	BR
MLS_S3	BR	BR	PT
MLS_S4	PT + BR	PT + BR	PT
MLS_S5	PT + BR	PT + BR	BR

Table 7.1: Models developed from scratch using subsets of the MLS dataset

The results of these experiments shown in Table 7.2 reveal that in any combination of the subsets the amount of data available is not enough to generate deep models with acceptable performance. It can also be concluded that the larger amounts of data produce better performances since the best results were achieved when training and testing the models using the largest subsets. Such can be seen in experiments MLS_S2 and MLS_S5 in which the models were trained and tested with the Brazilian Portuguese subsets.

Experiment	WER
MLS_S1	0.9952
MLS_S2	0.8156
MLS_S3	0.8842
MLS_S4	0.8900
MLS_S5	**0.8065**

Table 7.2: WER of models developed from scratch using the MLS dataset

The experiments of training models from scratch were also performed with the SpeechDat dataset. Regarding these experiments, the models were developed with SpeechDat's train and validation subsets.

As mentioned in Section 6.3.2, the SpeechDat dataset was divided into two "versions". Definitions 7.1 and Definition 7.2 define both versions of the dataset.

Definition 7.1: SpeechDat *with* noise describing tokens

The data pre-processing carried out in this iteration of the dataset only excluded audio recordings labelled as GARBAGE or OTHER, with transcriptions being **, empty or containing ~ or *.
The audio recordings which were kept were also re-encoded from A-LAW into WAV and re-sampled from 8 kHz to 16 kHz.

Definition 7.2: SpeechDat *without* noise describing tokens

The second iteration of the dataset has an additional pre-processing to Definition 7.1 in which the noise describing tokens, [sta], [spk], [fil] and [int], were removed from the transcriptions.

Table 7.3 presents the experiments carried out using the SpeechDat dataset as defined in Definition 7.1. Table 7.4 displays experiments performed using the SpeechDat dataset after an additional pre-processing step defined in Definition 7.2. .

Experiment	Train	Validation	Test	WER
SD_S1	SpeechDat	SpeechDat	SpeechDat	**0.3035**
SD_S2	SpeechDat	SpeechDat	ENG	0.9962
SD_S3	SpeechDat	SpeechDat	PT + BR	0.9173

Table 7.3: Models developed from scratch with the SpeechDat dataset as defined in Definition 7.1

Experiment	Train	Validation	Test	WER
SD_S4	SpeechDat	SpeechDat	SpeechDat	**0.1945**
SD_S5	SpeechDat	SpeechDat	ENG	0.9924
SD_S6	SpeechDat	SpeechDat	PT + BR	0.9055

Table 7.4: Models developed from scratch with the SpeechDat dataset as defined in Definition 7.2

The WER results yielded from these experiments show that models developed using SpeechDat's data perform better than models developed with the MLS dataset. This conclusion is true for both experiments with and without the additional pre-processing of the data. Upon the removal of noise and word pronunciation error tokens, it can be seen an increase in performance from experiment SD_S1 (0.3035) to experiment SD_S4 (0.1945).

7.2.3 Transfer learning

Deep learning models require large amounts of data to be able to generate good results and, as far as we know, there are no large public speech datasets for Portuguese. In contrast, English has a very large quantity of public speech data available due to the larger number of speakers and the majority of work in the area of speech recognition is developed for English. Considering the results from Section 7.2.2 it can be concluded that creating deep neural models from scratch would require more data than we have access to. Consequently, experiments using both the SpeechDat and MLS datasets have been developed using the transfer learning technique.

A large variety of subsets, regarding training, validation and testing sets, were used in these experiments. The "PT" and "BR" MLS subsets were again used separately and together. Furthermore, different mixes of SpeechDat and MLS subsets were also created. The 100 hours "clean" train set (ENG) from the LibriSpeech dataset was used as the control dataset for the English language. NVIDIA's NeMo provides

a model for the English language pre-trained with a dataset with ≃3300 hours of audio recordings[4]. This model was used as the starting point for the transfer learning experiments.

During the development of the transfer learning experiments it was observed that the values of the ROUGE-L metric were perfectly consistent with the WER values, *i.e.* the best WER (closer to 0) was also the best ROUGE-L (closer to 1). Hence, as these values are consistent across metrics, we focused our analysis only on the WER metric.

Table 7.5 shows the performance of the model pre-trained for the English language on the test subsets of MLS, LibriSpeech (ENG) and SpeechDat. These results will allow an understanding of how effective the transfer technique is in the following experiments, *i.e.* how much the pre-trained model adapts to/learns from data of different subsets.

Train/Validation	Test	WER
Pre-trained ENG	ENG	**0.0159**
	PT	0.9865
	BR	0.9863
	PT + BR	0.9863

Table 7.5: Pre-trained English model performance on the different test subsets

Additionally, to complement the previous experiment, the transfer technique was applied to the pre-trained English model using the MLS transfer subsets. Models were developed using the MLS transfer sets "PT", "BR" and "PT+BR" and were tested with the English test subset. Table 7.6 shows the results of this additional experiment. Results reveal good effectiveness of the transfer process, *i.e.* the pre-trained model updates towards the data of the transfer subsets, while performance decreases when testing with the English subset.

Train/Validation	Transfer	Test	WER
Pre-trained ENG	PT	ENG	**1.0238**
	BR	ENG	0.9905
	PT + BR	ENG	0.9922

Table 7.6: Performance on the English test subset on models created with transfer learning

The experiments on Tables 7.7, 7.8 and 7.9 are similar to scratch experiments carried out in Section 7.2.2. The goal of these experiments is to make a comparison between the performance of models developed from scratch and models created using the transfer learning approach.

Comparing both scratch and transfer learning approaches some conclusions can be made regarding the quantity of data used during the training phase and how it affects performance.

[4] https://developer.nvidia.com/blog/jump-start-training-for-speech-recognition-models-with-nemo/

Train/Validation	Transfer	Test	WER
Pre-trained ENG	PT	PT	1.0164
	PT	BR	0.9722
	BR	PT	0.7075
	BR	BR	0.5139
	PT + BR	BR	**0.5025**
	PT + BR	PT + BR	0.5083

Table 7.7: Performance of models developed using transfer learning with the MLS subsets

Train/Validation	Transfer	Test	WER
Pre-trained ENG	SpeechDat	SpeechDat	**0.1603**
	SpeechDat	ENG	1.0186
	SpeechDat	PT + BR	0.7912

Table 7.8: Performance of models developed using SpeechDat as transferring set

Train/Validation	Transfer	Test	WER
Pre-trained ENG	SpeechDat	SpeechDat	**0.0557**
	SpeechDat	ENG	1.006
	SpeechDat	PT + BR	0.7680

Table 7.9: Performance of models developed using SpeechDat as transferring set after data processing

1. The amount of data used during the training phase strongly impacts the model's performance; transfer learning improves the performance from 0.1945 to 0.0557
2. It has been found that pre-trained models successfully adapt to new data by moving away from the data they were originally trained on.

Comparing both paradigms, from scratch and transfer learning, it can be concluded that models perform better when developed with more data, *i.e.* using transfer learning. Although transferring knowledge from previously developed models achieves better scores, it is observable that the transfer set still needs to have a considerable amount of data. This conclusion is supported by the following:

- when using smaller quantities of data, *i.e.* MLS subsets, the best results were achieved when using combinations with the largest subset ("BR")
- models that yielded the best results were developed with the largest subsets available, *i.e.* SpeechDat subsets

The results of these experiments are as well a reinforcement to the statement that better data quality can be as important as data quantity and algorithm fine-tuning. Models developed with the SpeechDat dataset have been shown an increase in performance after additional data processing even without any model fine-tuning or additional data added.

Conclusions from the previous experiments led to another batch of experiments using transfer learning whose goal is to evaluate the impact of data quality and quantity on the performance of the models. These experiments consisted in using mixes of the SpeechDat and the MLS[5] transfer and validation subsets. As can be seen in Table 7.10, the proportion of each subset is defined by a linear variation between 0% and 100% with a step of 25%. This is done so that the number of instances remains constant across all mixes, 107158 for training and 22960 for validation. These proportions were defined in order for the total number of audio recordings to be equal in all training and validation mixes respectively.

As explored in Chapter 3, the MLS and SpeechDat datasets differ in the quantity of data, audio recording sources and consequently data quality. Combining both datasets makes it possible to investigate how the data quality and the quantity, *i.e.*, whether collected through an open source crowdsourcing (MLS) or in a controlled environment (SpeechDat), affects the performance of automatic speech recognition models.

[5]In these experiments "MLS" represents the "PT + BR" both training and testing subsets

MIX ID	Transfer		Validation	
	MLS	SpeechDat	MLS	SpeechDat
0.00	0	107158	0	22960
0.25	9374	97784	206	22754
0.50	18749	88409	413	22547
0.75	28123	79035	619	22341
1.00	37498	69660	826	22134

Table 7.10: Audio instances from each dataset used on each training and validation mix

Test set	Instances
SpeechDat	22961
ENG	28539
MLS	906
SpeechDat + MLS	23867

Table 7.11: Audio instances from each test set used

For each mix of the datasets, three models were developed and evaluated. The results of these experiments on each of the mixes can be seen on Tables 7.12, 7.13, 7.14, 7.15 and 7.16. Table 7.17 presents a summarised view of the average performance of each mix.

The performance of the models developed in these experiments was evaluated using the previously used test sets, SpeechDat, MLS ("PT + BR") and LibriSpeech (ENG). Additionally, a full combination of the SpeechDat and MLS test sets (SpeechDat + MLS) was also used to evaluate the model's performance. Table 7.11 displays the number of audio instances present in each of the mentioned test sets.

The English set once again allows to ensure that the transfer learning process is effective, *i.e.* the pre-trained model is shifting towards the new data. The other three remaining test sets, SpeechDat, MLS and SpeechDat + MLS, grant the possibility to study how models generalise based on the proportions of data used in the transfer learning process.

In order, the mixes present in Table 7.10 start with a high number of instances from SpeechDat and a low number from MLS and end with a more balanced proportion of both relative to their original sizes. Following the mix order present in the previously mentioned table to look at the results in Tables 7.12, 7.13, 7.14, 7.15 and 7.16 the following conclusions can be taken regarding the results from these experiments.

After a general analysis of the results, some patterns are observable concerning the performance of each test set on the models developed for each mix. The only outlier is the results obtained on the third model of the second mix (MIX ID 0.25, Table 7.13)

It can be concluded that the SpeechDat dataset has the greatest impact on the performance of the models.

Test set	WER			
	Model 1	Model 2	Model 3	Average
SpeechDat	0.0534	0.0503	0.0503	0.0513
ENG	0.9982	1.0019	1.0040	1.0014
MLS	0.7670	0.7616	0.7759	0.7682
SpeechDat + MLS	0.1903	0.1868	0.1966	0.1912

Table 7.12: Individual and average performances of the models developed with MIX 0.00

Test set	WER			
	Model 1	Model 2	Model 3	Average
SpeechDat	0.0594	0.0630	0.2739	0.1321
ENG	0.9954	0.9940	1.0011	0.9969
MLS	0.4670	0.4723	0.7601	0.5665
SpeechDat + MLS	0.1377	0.1415	0.3672	0.2155

Table 7.13: Individual and average performances of the models developed with MIX 0.25

Test set	WER			
	Model 1	Model 2	Model 3	Average
SpeechDat	0.0560	0.0603	0.0578	0.0581
ENG	1.0006	0.9986	1.0048	1.0013
MLS	0.3814	0.3974	0.3965	0.3918
SpeechDat + MLS	0.1185	0.1250	0.1228	0.1221

Table 7.14: Individual and average performances of the models developed with MIX 0.50

Test set	WER			
	Model 1	Model 2	Model 3	Average
SpeechDat	0.0666	0.0646	0.0690	0.0667
ENG	0.9958	0.9939	0.9932	0.9943
MLS	0.3600	0.3499	0.3624	0.3574
SpeechDat + MLS	0.1229	0.1194	0.1253	0.1225

Table 7.15: Individual and average performances of the models developed with MIX 0.75

Test set	WER			
	Model 1	Model 2	Model 3	Average
SpeechDat	0.0669	0.0721	0.0817	0.0735
ENG	0.9930	0.9951	0.9966	0.9949
MLS	0.3140	0.3314	0.3464	0.3306
SpeechDat + MLS	0.1143	0.1218	0.1332	0.1231

Table 7.16: Individual and average performances of the models developed with MIX 1.00

Test set	Average WER				
	MIX ID 0.00	MIX ID 0.25	MIX ID 0.50	MIX ID 0.75	MIX ID 1.00
SpeechDat	0.0513	0.1321	0.0581	0.0667	0.0735
ENG	1.0014	0.9969	1.0013	0.9943	0.9949
MLS	0.7682	0.5665	0.3918	0.3574	0.3306
SpeechDat + MLS	0.1912	0.2155	0.1221	0.1225	0.1231

Table 7.17: Average performance of the models of each MIX

The performance on the SpeechDat test set slightly decreased from mix to mix, having a variation on the average performance of 0.0222 from the first (MIX ID 0.00, Table 7.12) to the last mix (MIX ID 1.00, Table 7.16) and a variance of 0.0322 among the five mixes.

Although the MLS dataset is smaller, throughout the mix experiments, the increase in the number of instances used is reflected in the gain of performance over the MLS and SpeechDat + MLS test sets throughout the different mixes. Nonetheless, in spite of achieving better performances over these test sets, the addition of these instances doesn't show a negative impact on the performance over the SpeechDat and LibriSpeech (ENG) test sets.

Concluding these experiments' discussion, Table 7.18 shows the gain/loss of performance over the SpeechDat, MLS and SpeechDat + MLS test sets between the first (MIX ID 00.00) and last mix (MIX ID 1.00). The mixture of these two sources of data has shown an overall increase in performance, leading to the following conclusions:

- the quantity of data is undoubtedly essential for building deep neural models, either from scratch or using transfer learning
- the larger the amount of data from a source used in the train or transfer process, the better the performance over the test set from the same source
- data quality performs a major role in the model's performance since the dataset with better data quality (in this work was the SpeechDat dataset) yields the best performance

Test set	Average WER		WER difference
	MIX ID 0.00	MIX ID 1.00	
SpeechDat	0.0513	0.0735	-0.0222
ENG	1.0014	0.9949	0.0065
MLS	0.7682	0.3306	0.4376
SpeechDat + MLS	0.1912	0.1231	0.0681

Table 7.18: WER difference from the first (MIX ID 0.00) and last mix (MIX ID 1.00)

8

Conclusion and Future Work

This final Chapter describes:

- 8.1 - **Conclusion** - an overall conclusion of what was accomplished in the present work
- 8.2 - **Future work** - future work to be done in order to improve the developed system and yield better results

8.1 Conclusion

In Section 1.3 it was proposed the development of an automatic speech recognition model for the Portuguese language using deep learning techniques. After analysing the state-of-the-art literature and automatic speech recognition frameworks, end-to-end deep neural networks were defined as the model to be used in the current work. The core strategy of this work had the goal to develop such models following a data-centric methodology.

The system was developed using a pipeline. When assembled, this pipeline made the workflow of carrying the current work's experiments smooth and simple by providing simple methods for changing between experiments. The use of independent services for each stage of the pipeline also made debugging easier when encountering difficulties, such as the attempt to make use of multi-node computing.

The NVIDIA NeMo framework enabled the implementation of the QuartzNet15x5 architecture based on 1D time-channel separable convolutions. In combination with the data-centric methodology, this architecture allowed the development of the main contribution of this dissertation: an automatic speech recognition system for the European Portuguese languages which achieves state-of-the-art results. The best model achieves a performance of $WER = 0.0503$.

8.2 Future work

The present work presents very good results close to state-of-the-art. Future works in automatic speech recognition for the Portuguese language have room to improve the yielded results. Among others, some approaches that can be explored in future works and are worth mentioning are:

- algorithm fine-tuning, which was not carried out in this work
- a finer look and implementation of data-centric methodologies, which revealed to improve model's performance
- the implementation of language models that usually significantly improve the accuracy of the speech recogniser [Han17]

Regarding algorithm fine-tuning, possible paths for future work could be tuning parameters of components such as data augmenters, optimisers and schedulers. Investigating how varying these impacts the augmented data and learning rate and hence the model's performance.

A possible path to follow based on data-centric methodologies for future works could be focused on approaches to integrate additional datasets and/or exploring different and/or additional data pre-processing.

Language models model the likelihood of a word in a sequence of words. Although such models tend to perform better with other models than with CTC-trained models [Han17], separately trained language models might improve the model's performance by decreasing the number of miss-spelt or miss-placed words.

Bibliography

[AAB+] Dario Amodei, Rishita Anubhai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Jingdong Chen, Mike Chrzanowski, Adam Coates, Greg Diamos, Erich Elsen, Jesse Engel, Linxi Fan, Christopher Fougner, Tony Han, Awni Hannun, Billy Jun, Patrick Legresley, Libby Lin, Sharan Narang, Andrew Ng, Sherjil Ozair, Ryan Prenger, Jonathan Raiman, Sanjeev Satheesh, David Seetapun, Shubho Sengupta, Yi Wang, Zhiqian Wang, Chong Wang, Bo Xiao, Dani Yogatama, Jun Zhan, and Zhenyao Zhu. Deep Speech 2: End-to-End Speech Recognition in English and Mandarin Baidu Research-Silicon Valley AI Lab *.

[AAB+15] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, Xiaoqiang Zheng, and Google Research. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. 2015.

[AdLDCA20] Thales Aguiar de Lima and Márjory Da Costa-Abreu. A survey on automatic speech recognition systems for Portuguese language and its variations. *Computer Speech and Language*, 62, 7 2020.

[AHMJ+14] Ossama Abdel-Hamid, Abdel Rahman Mohamed, Hui Jiang, Li Deng, Gerald Penn, and Dong Yu. Convolutional neural networks for speech recognition. *IEEE Transactions on Audio, Speech and Language Processing*, 22(10):1533–1545, 10 2014.

[Ami21] José Manuel Amigo. Data Mining, Machine Learning, Deep Learning, Chemometrics Definitions, Common Points and Trends (Spoiler Alert: VALIDATE your models!). *Brazilian Journal of Analytical Chemistry*, 8(32):45–61, 2021.

[Ban19] Siddhant Bansal. Decoding Connectionist Temporal Classification | Siddhant's Scratch Book, 2019.

[BBKS14] Laurent Besacier, Etienne Barnard, Alexey Karpov, and Tanja Schultz. Automatic speech recognition for under-resourced languages: A survey. 2014.

[BJM83] Lalit R Bahl, Frederick Jelinek, and Robert L Mercer. A Maximum Likelihood Approach to Continuous Speech Recognition. *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE*, 5(2), 1983.

[BKD21] Ruurd Buijs, Thomas Koch, and Elenna Dugundji. Applying transfer learning and various ANN architectures to predict transportation mode choice in Amsterdam. *Procedia Computer Science*, 184:532–540, 1 2021.

[BM94] Hervé A. Bourlard and Nelson Morgan. Connectionist Speech Recognition. *Connectionist Speech Recognition: A Hybrid Approach*, 1994.

[Bri21] James Briggs. Measure NLP Accuracy With ROUGE | Towards Data Science, 3 2021.

[BZP+19] Eric Breck, Marty Zinkevich, Neoklis Polyzotis, Steven Whang, and Sudip Roy. Data Validation for Machine Learning, 2019.

[CKF] Ronan Collobert, Koray Kavukcuoglu, and Clément Farabet. Torch7: A Matlab-like Environment for Machine Learning.

[Co15] François Chollet and others. Keras. https://keras.io, 2015.

[CPS] Ronan Collobert, Christian Puhrsch, and Gabriel Synnaeve. Wav2Letter: an End-to-End ConvNet-based Speech Recognition System.

[CPS06] K. Chellapilla, Sidd Puri, and P. Simard. High Performance Convolutional Neural Networks for Document Processing. *undefined*, 2006.

[DBB52] K. H. Davis, R. Biddulph, and S. Balashek. Automatic Recognition of Spoken Digits. *undefined*, 24(6):637–642, 1952.

[DSH13] George E. Dahl, Tara N. Sainath, and Geoffrey E. Hinton. Improving deep neural networks for LVCSR using rectified linear units and dropout. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, pages 8609–8613, 10 2013.

[DT17] Terrance DeVries and Graham W. Taylor. Improved Regularization of Convolutional Neural Networks with Cutout. 8 2017.

[Fay16] Haytham M Fayek. Speech Processing for Machine Learning: Filter banks, Mel-Frequency Cepstral Coefficients (MFCCs) and What's In-Between, 2016.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[GCF+06] Alex Graves, Alex@idsia Ch, Santiago Fernández, Faustino Gomez, Jürgen Schmidhuber, and Juergen@idsia Ch. Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks. 2006.

[GCH+19] Boris Ginsburg, Patrice Castonguay, Oleksii Hrinchuk, Oleksii Kuchaiev, Vitaly Lavrukhin, Ryan Leary, Jason Li, Huyen Nguyen, Yang Zhang, and Jonathan M. Cohen. Stochastic Gradient Methods with Layer-wise Adaptive Moments for Training of Deep Networks. 5 2019.

[GCO+21] Lucas Rafael Stefanel Gris, Edresson Casanova, Frederico Santos de Oliveira, Anderson da Silva Soares, and Arnaldo Candido-Junior. Desenvolvimento de um modelo de reconhecimento de voz para o Português Brasileiro com poucos dados utilizando o Wav2vec 2.0. *Anais do Brazilian e-Science Workshop (BreSci)*, pages 129–136, 7 2021.

[GLF+] Yunhui Guo, Yandong Li, Rogerio Feris, Liqiang Wang, and Tajana Rosing. Depthwise Convolution is All You Need for Learning Multiple Visual Domains.

[GMH13] Alex Graves, Abdel-Rahman Mohamed, and Geoffrey Hinton. SPEECH RECOGNITION WITH DEEP RECURRENT NEURAL NETWORKS. 2013.

[GPCB21] Alexandru-Lucian Georgescu, Alessandro Pappalardo, Horia Cucu, and Michaela Blott. Performance vs. hardware requirements in state-of-the-art automatic speech recognition. *EURASIP Journal on Audio*, 2021:28, 2021.

[GS05] Alex Graves and Jürgen Schmidhuber. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Networks*, 18(5-6):602–610, 7 2005.

[GTK22] Maryna Garan, Khaoula Tidriri, and Iaroslav Kovalenko. A Data-Centric Machine Learning Methodology: Application on Predictive Maintenance of Wind Turbines. *Energies 2022, Vol. 15, Page 826*, 15(3):826, 1 2022.

[Han17] Awni Hannun. Sequence Modeling with CTC. *Distill*, 2(11):e8, 11 2017.

[HCC+] Annika Hämäläinen, Hyongsil Cho, Sara Candeias, Thomas Pellegrini, Alberto Abad, Michael Tjalve, Isabel Trancoso, and Miguel Sales Dias. Automatically Recognising European Portuguese Children's Speech: Pronunciation Patterns Revealed by an Analysis of ASR Errors.

[HDY+12] Geoffrey Hinton, Li Deng, Dong Yu, George Dahl, Abdel Rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara Sainath, and Brian Kingsbury. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

[HES00] Hynek Hermansky, Daniel P.W. Ellis, and Sangita Sharma. Tandem connectionist feature extraction for conventional HMM systems. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, 3:1635–1638, 2000.

[HJB84] Michael T. Heideman, Don H. Johnson, and C. Sidney Burrus. Gauss and the History of the Fast Fourier Transform. *IEEE ASSP Magazine*, 1(4):14–21, 1984.

[HLVDMW17] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely Connected Convolutional Networks. 2017.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 11 1997.

[JG03] Moris Jette and Mark Grondona. SLURM: Simple Linux Utility for Resource Management | Request PDF. 2003.

[KBG+19] Samuel Kriman, Stanislav Beliaev, Boris Ginsburg, Jocelyn Huang, Oleksii Kuchaiev, Vitaly Lavrukhin, Ryan Leary, Jason Li, and Yang Zhang. QUARTZNET: DEEP AUTOMATIC SPEECH RECOGNITION WITH 1D TIME-CHANNEL SEPARABLE CONVOLUTIONS. 2019.

[KLGS17] Karlis Kanders, Tom Lorimer, Florian Gomez, and Ruedi Stoop. Frequency sensitivity in mammalian hearing from a fundamental nonlinear physics model of the inner ear. *Scientific Reports 2017 7:1*, 7(1):1–8, 8 2017.

[KLN+19] Oleksii Kuchaiev, Jason Li, Huyen Nguyen, Oleksii Hrinchuk, Ryan Leary, Boris Ginsburg, Samuel Kriman, Stanislav Beliaev, Vitaly Lavrukhin, Jack Cook, Patrice Castonguay, Mariya Popova, Jocelyn Huang, and Jonathan M Cohen. NeMo: a toolkit for building AI applications using Neural Modules. 2019.

[Lee94] Chin Hui Lee. Maximum a Posteriori Estimation for Multivariate Gaussian Mixture Observations of Markov Chains. *IEEE Transactions on Speech and Audio Processing*, 2(2):291–298, 1994.

[LGTB97] Steve Lawrence, C. Lee Giles, Ah Chung Tsoi, and Andrew D. Back. Face recognition: A convolutional neural-network approach. *IEEE Transactions on Neural Networks*, 8(1):98–113, 1997.

[LH] Ilya Loshchilov and Frank Hutter. SGDR: STOCHASTIC GRADIENT DESCENT WITH WARM RESTARTS.

[LHB04] Yann LeCun, Fu Jie Huang, and Léon Bottou. Learning methods for generic object recognition with invariance to pose and lighting. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2, 2004.

[Lin04] Chin-Yew Lin. ROUGE: A Package for Automatic Evaluation of Summaries. pages 74–81, 2004.

[LLG+19] Jason Li, Vitaly Lavrukhin, Boris Ginsburg, Ryan Leary, Oleksii Kuchaiev, Jonathan M Cohen, Huyen Nguyen, and Ravi Teja Gadde. Jasper: An End-to-End Convolutional Neural Acoustic Model. 2019.

[MAP+] Hugo Meinedo, Alberto Abad, Thomas Pellegrini, João Neto, and Isabel Trancoso. The L2F Broadcast News Speech Recognition System.

[MCNT03] Hugo Meinedo, Diamantino Caseiro, João Neto, and Isabel Trancoso. AUDIMUS.media: A Broadcast News Speech Recognition System for the European Portuguese Language. Technical report, 2003.

[MQ17] Igor Macedo Quintanilha. END-TO-END SPEECH RECOGNITION APPLIED TO BRAZILIAN PORTUGUESE USING DEEP LEARNING. 2017.

[MVKK] Gonçalo Mordido, Matthijs Van Keirsbilck, and Alexander Keller. Compressing 1D Time-Channel Separable Convolutions using Sparse Random Ternary Matrices.

[PCPK15] Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur. LIBRISPEECH: AN ASR CORPUS BASED ON PUBLIC DOMAIN AUDIO BOOKS. 2015.

[PDC15] Dimitri Palaz, Mathew Magimai Doss, and Ronan Collobert. Analysis of CNN-based Speech Recognition System using Raw Speech as Input. 2015.

[PGM+19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury Google, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf Xamla, Edward Yang, Zach Devito, Martin Raison Nabla, Alykhan Tejani, Sasank Chilamkurthy, Qure Ai, Benoit Steiner, Lu Fang Facebook, Junjie Bai Facebook, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. 2019.

[PHBDM+13] Thomas Pellegrini, Annika Hämäläinen, Philippe Boula De Mareüil, Michael Tjalve, Isabel Trancoso, Sara Candeias, Miguel Sales Dias, and Daniela Braga. A corpus-based study of elderly and young speakers of European Portuguese: acoustic correlates and their impact on speech recognition performance. 2013.

[PXS+20] Vineel Pratap, Qiantong Xu, Anuroop Sriram, Gabriel Synnaeve, and Ronan Collobert. MLS: A LARGE-SCALE MULTILINGUAL DATASET FOR SPEECH RESEARCH A PREPRINT. Technical report, 2020.

[QNB20] Igor Macedo Quintanilha, Sergio Lima Netto, and Luiz Wagner Pereira Biscainho. An open-source end-to-end ASR system for Brazilian Portuguese using DNNs built from newly assembled corpora. *Journal of Communication and Information Systems*, 35(1):230–242, 9 2020.

[Rab97] Lawrence R. Rabiner. Applications of speech recognition in the area of telecommunications. *IEEE Workshop on Automatic Speech Recognition and Understanding Proceedings*, pages 501–510, 1997.

[RG] Lucas Rafael and Stefanel Gris. Brazilian Portuguese Speech Recognition Using Wav2vec 2.0. Technical report.

[RKX+22] Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine Mcleavey, and Ilya Sutskever. Robust Speech Recognition via Large-Scale Weak Supervision. 2022.

[RWL+16] Sherry Ruan, Jacob O Wobbrock, Kenny Liou, Andrew Ng, and James Landay. Speech Is 3x Faster than Typing for English and Mandarin Text Entry on Mobile Devices. 2016.

[SBS05] Dave Steinkraus, Ian Buck, and Patrice Y. Simard. Using GPUs for machine learning algorithms. *undefined*, 2005:1115–1120, 2005.

[SKSKA+17] George Saon, Gakuto Kurata, Tom Sercu Kartik Audhkhasi, Samuel Thomas, Dimitrios Dimitriadis Xiaodong Cui, Bhuvana Ramabhadran, Michael Picheny, Lynn-Li Lim, Bergul Roomi, and Phil Hall Appen. English Conversational Telephone Speech Recognition by Humans and Machines. Technical report, 2017.

[SMKR13] Tara N. Sainath, Abdel Rahman Mohamed, Brian Kingsbury, and Bhuvana Ramabhadran. Deep convolutional neural networks for LVCSR. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, pages 8614–8618, 10 2013.

[Spe] Speaker Identification Using Pitch and MFCC - MATLAB & Simulink.

[ST13] Urmila Shrawankar and V M Thakare. Techniques for Feature Extraction In Speech Recognition System : A Comparative Study. 5 2013.

[SVN05] S. S. Stevens, J. Volkmann, and E. B. Newman. A Scale for the Measurement of the Psychological Magnitude Pitch. *The Journal of the Acoustical Society of America*, 8(3):185, 6 2005.

[TKEY] Natalia Tomashenko, Yuri Khokhlov, Yannick Estève, and Estève Yannick. Exploring Gaussian mixture model framework for speaker adaptation of deep neural network acoustic models Exploring Gaussian mixture model framework for speaker adaptation of deep neural network acoustic models Exploring Gaussian mixture model framework for speaker adaptation of deep neural network acoustic models. Technical report.

[Tob05] Richard Tobin. Automatic Speech Recognition Implementations in Healthcare. 2005.

[VBS+17] Ashish Vaswani, Google Brain, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. 2017.

[Wan03] Sun-Chong Wang. Artificial Neural Network. *Interdisciplinary Computing in Java Programming*, pages 81–100, 2003.

[Wha] What is Artificial Intelligence (AI) ? | IBM.

[WHH+89] Alexander Waibel, Toshiyuki Hanazawa, Geoffrey Hinton, Kiyohiro Shikano, and Kevin J. Lang. Phoneme Recognition Using Time-Delay Neural Networks. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 37(3):328–339, 1989.

[WŠW+21] Hannes Westermann, Jaromír Šavelka, Vern R. Walker, Kevin D. Ashley, and Karim Benyekhlef. Data-Centric Machine Learning: Improving Model Performance and Understanding Through Dataset Analysis. *Frontiers in Artificial Intelligence and Applications*, 346:54–57, 12 2021.

[XSPMLCdS+] Matheus Xavier Sampaio, Regis Pires Magalhães, Ticiana Linhares Coelho da Silva, Lívia Almada Cruz, Davi Romero de Vasconcelos, José Antônio Fernandes de Macêdo, and Marianna Gonçalves Fontenele Ferreira. Evaluation of Automatic Speech Recognition Systems. Technical report.

[Yad19] Omry Yadan. Hydra - A framework for elegantly configuring complex applications. Github, 2019.

[YCBL] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks?

[ZA] Yaxin Zhang and Mike Alder. USING GAUSSIAN MIXTURE MODELING IN SPEECH RECOGNITION.

[ZRM+13] M. D. Zeiler, M. Ranzato, R. Monga, M. Mao, K. Yang, Q. V. Le, P. Nguyen, A. Senior, V. Vanhoucke, J. Dean, and G. E. Hinton. On rectified linear units for speech processing. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, pages 3517–3521, 10 2013.

[ZXL+] Neil Zeghidour, Qiantong Xu, Vitaliy Liptchinsky, Nicolas Usunier, Gabriel Synnaeve, and Ronan Collobert. Fully Convolutional Speech Recognition.

[ZYL+18] W. J. Zhang, Guosheng Yang, Yingzi Lin, Chunli Ji, and Madan M. Gupta. On Definition of Deep Learning. *World Automation Congress Proceedings*, 2018-June:232–236, 8 2018.

A

Developed software

A.1 Train script

```
# NeMo"s "core" package
import nemo
import torch
import hydra
from omegaconf import DictConfig, OmegaConf

# NeMo"s ASR collection - this collections contains complete ASR models
    and
# building blocks (modules) for ASR
import nemo.collections.asr as nemo_asr

import pytorch_lightning as pl  # Used for training
```

```
12
13
14 @hydra.main(config_path="./configs/")
15 def main(cfg: DictConfig):
16
17     trainer = pl.Trainer(**cfg.trainer)
18     model = nemo_asr.models.EncDecCTCModel(cfg=cfg.model, trainer=trainer
       )
19
20     model.maybe_init_from_pretrained_checkpoint(cfg)
21
22     # Train the model
23     trainer.fit(model)
24
25     model.save_to(f"./models/{cfg.name}.nemo")
26
27
28 if __name__ == "__main__":
29     main()
```

Listing A.1: Script developed for model training

A.2 Test script

```
1  import argparse
2  import json
3
4  import nemo
5  import nemo.collections.asr as nemo_asr
6  from nemo.collections.asr.metrics.wer import word_error_rate
7  from torchmetrics import BLEUScore, WordErrorRate
8  from torchmetrics.text.rouge import ROUGEScore
9
10
11 BATCH_SIZE = 32
12
13
14 def calculate_wer(args, model):
15
16     test_manifest = args.test_path
17
18     ## Calculate WER between hypothesis and ground_truth
19     original_text = []
20     audio_filepaths = []
21
22     with open(test_manifest) as f:
```

```
23          for line in f:
24              json_line = json.loads(line)
25              original_text.append(json_line['text'])
26              audio_filepaths.append(json_line['audio_filepath'])
27
28      transcribes = model.transcribe(paths2audio_files=audio_filepaths,
            batch_size=args.batch_size)
29
30      if args.show_transcribes:
31          print(transcribes)
32
33      # Calculate WER
34      wer = word_error_rate(hypotheses=transcribes, references=
            original_text, use_cer=False)
35      print(f"\nMETRICS:")
36      print(f"\tWER={wer}")
37
38      print(f"TORCH METRICS:")
39
40      metric = WordErrorRate()
41      print(f"\tWER={metric(transcribes, original_text)}")
42
43      metric = BLEUScore()
44      print(f"\tBLEUEScore={metric(transcribes, original_text)}")
45
46      metric = ROUGEScore()
47      print(f"\tROUGEScore={metric(transcribes, original_text)}")
48
49
50  def transcribe(args, model):
51
52      print(args.audio_path)
53
54      transcribes = model.transcribe(paths2audio_files=[args.audio_path],
            batch_size=args.batch_size)
55
56      print(transcribes)
57
58
59  def parse_args() -> list:
60      parser = argparse.ArgumentParser()
61
62      parser.add_argument("--test_path", type=str)
63      parser.add_argument("--model_name", type=str)
64
65      parser.add_argument("--transcribe", action="store_true")
66      parser.add_argument("--show_transcribes", type=bool)
67      parser.add_argument("--audio_path", type=str)
68
```

```
    parser.add_argument("--batch_size", type=int)

    args = parser.parse_args()

    return args


if __name__ == '__main__':

    args = parse_args()

    save_path = f"{args.model_name}" if args.model_name.endswith(".nemo")
        else f"{args.model_name}.nemo"

    # To load a model from disk
    model = nemo_asr.models.EncDecCTCModel.restore_from(restore_path=f"{
       save_path}")

    if args.transcribe:
        transcribe(args, model)
        exit(1)

    calculate_wer(args, model)
```

Listing A.2: Script developed for model evaluation

A.3 Configuration example

```
name: &name "MODEL_NAME"

model:
  sample_rate: &sample_rate 16000
  repeat: &repeat 5
  dropout: &dropout 0.0
  separable: &separable true
  batch_size: &batch_size 32
  num_workers: &num_workers 256
  labels: &labels [" ", "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", "ç", "à", "á",
                   "â", "ã", "é", "ê", "í", "ó", "ô", "õ", "ú", "-", "'"]

  train_ds:
    manifest_filepath: "train_manifest.json"
    sample_rate: *sample_rate
    labels: *labels
```

```
    batch_size: *batch_size
    trim_silence: True
    max_duration: 16.7
    shuffle: True
    # tarred datasets
    is_tarred: false
    tarred_audio_filepaths: null
    shuffle_n: 2048
    # bucketing params
    bucketing_strategy: "synced_randomized"
    bucketing_batch_size: null
    num_workers: *num_workers

  validation_ds:
    manifest_filepath: "validation_manifest.json"
    sample_rate: *sample_rate
    labels: *labels
    batch_size: *batch_size
    shuffle: False
    num_workers: *num_workers

  test_ds:
    manifest_filepath: "test_manifest.json"
    sample_rate: *sample_rate
    labels: *labels
    batch_size: *batch_size
    shuffle: False
    num_workers: *num_workers

  preprocessor:
    _target_: nemo.collections.asr.modules.
       AudioToMelSpectrogramPreprocessor
    normalize: "per_feature"
    window_size: 0.02
    sample_rate: *sample_rate
    window_stride: 0.01
    window: "hann"
    features: &n_mels 64
    n_fft: 512
    frame_splicing: 1
    dither: 0.00001

  spec_augment:
    _target_: nemo.collections.asr.modules.SpectrogramAugmentation
    rect_freq: 50
    rect_masks: 5
    rect_time: 100

  encoder:
```

```
    _target_: nemo.collections.asr.modules.ConvASREncoder
    feat_in: *n_mels
    activation: relu
    conv_mask: true

    jasper:
    - dilation: [1]
      dropout: *dropout
      filters: 256
      kernel: [33]
      repeat: 1
      residual: false
      separable: *separable
      stride: [2]

    - dilation: [1]
      dropout: *dropout
      filters: 256
      kernel: [33]
      repeat: *repeat
      residual: true
      separable: *separable
      stride: [1]

    - dilation: [1]
      dropout: *dropout
      filters: 256
      kernel: [33]
      repeat: *repeat
      residual: true
      separable: *separable
      stride: [1]

    - dilation: [1]
      dropout: *dropout
      filters: 256
      kernel: [33]
      repeat: *repeat
      residual: true
      separable: *separable
      stride: [1]

    - dilation: [1]
      dropout: *dropout
      filters: 256
      kernel: [39]
      repeat: *repeat
      residual: true
      separable: *separable
```

```
    stride: [1]

  - dilation: [1]
    dropout: *dropout
    filters: 256
    kernel: [39]
    repeat: *repeat
    residual: true
    separable: *separable
    stride: [1]

  - dilation: [1]
    dropout: *dropout
    filters: 256
    kernel: [39]
    repeat: *repeat
    residual: true
    separable: *separable
    stride: [1]

  - dilation: [1]
    dropout: *dropout
    filters: 512
    kernel: [51]
    repeat: *repeat
    residual: true
    separable: *separable
    stride: [1]

  - dilation: [1]
    dropout: *dropout
    filters: 512
    kernel: [51]
    repeat: *repeat
    residual: true
    separable: *separable
    stride: [1]

  - dilation: [1]
    dropout: *dropout
    filters: 512
    kernel: [51]
    repeat: *repeat
    residual: true
    separable: *separable
    stride: [1]

  - dilation: [1]
    dropout: *dropout
```

```
    filters: 512
    kernel: [63]
    repeat: *repeat
    residual: true
    separable: *separable
    stride: [1]

  - dilation: [1]
    dropout: *dropout
    filters: 512
    kernel: [63]
    repeat: *repeat
    residual: true
    separable: *separable
    stride: [1]

  - dilation: [1]
    dropout: *dropout
    filters: 512
    kernel: [63]
    repeat: *repeat
    residual: true
    separable: *separable
    stride: [1]

  - dilation: [1]
    dropout: *dropout
    filters: 512
    kernel: [75]
    repeat: *repeat
    residual: true
    separable: *separable
    stride: [1]

  - dilation: [1]
    dropout: *dropout
    filters: 512
    kernel: [75]
    repeat: *repeat
    residual: true
    separable: *separable
    stride: [1]

  - dilation: [1]
    dropout: *dropout
    filters: 512
    kernel: [75]
    repeat: *repeat
    residual: true
```

```
      separable: *separable
      stride: [1]

    - dilation: [2]
      dropout: *dropout
      filters: 512
      kernel: [87]
      repeat: 1
      residual: false
      separable: *separable
      stride: [1]

    - dilation: [1]
      dropout: *dropout
      filters: &enc_filters 1024
      kernel: [1]
      repeat: 1
      residual: false
      stride: [1]

  decoder:
    _target_: nemo.collections.asr.modules.ConvASRDecoder
    feat_in: *enc_filters
    num_classes: 0
    vocabulary: *labels

  optim:
    name: novograd
    # _target_: nemo.core.optim.optimizers.Novograd
    lr: .01
    # optimizer arguments
    betas: [0.8, 0.5]
    weight_decay: 0.001

    # scheduler setup
    sched:
      name: CosineAnnealing

      # pytorch lightning args
      # monitor: val_loss
      # reduce_on_plateau: false

      # Scheduler params
      warmup_steps: null
      warmup_ratio: null
      min_lr: 0.0
      last_epoch: -1

trainer:
```

```
  gpus: -1 # number of gpus
  max_epochs: 100
  max_steps: -1 # computed at runtime if not set
  num_nodes: 1
  strategy: ddp
  accumulate_grad_batches: 1
  enable_checkpointing: False  # Provided by exp_manager
  logger: False  # Provided by exp_manager
  log_every_n_steps: 1  # Interval of logging.
  val_check_interval: 1.0  # Set to 0.25 to check 4 times per epoch, or
     an int for number of iterations


init_from_nemo_model:
  model0:
    path: "/data/asr-pt/nemo_models/QuartzNet15x5Base-En.nemo"
    exclude: ["decoder"]

exp_manager:
  exp_dir: null
  name: *name
  create_tensorboard_logger: False
  create_checkpoint_callback: False
  checkpoint_callback_params:
    monitor: "val_wer"
    mode: "min"
  create_wandb_logger: False
  wandb_logger_kwargs:
    name: null
    project: null

hydra:
  run:
    dir: .
  job_logging:
    root:
      handlers: null
```

Listing A.3: NeMo example configuration

A.4 Makefile

```
# Variables used for the python script commands
TEST_PATH = ./processed/altice_test.json

# Audio directory for single or multiple transcription
AUDIO_PATH = ./audio_to_test

# Default model name to train on the SpeechDat ds
```

```
MODEL_NAME = "PT_PT"

# Config path of model specs
CONFIG_NAME = "config"

# Specs for training
BATCH_SIZE = 32

# Different directory variables must be defined in the make command

# Trains model and outputs to a .nemo file
train: nemo_train.py
        @python3 nemo_train.py --config-name $(CONFIG_NAME)

# Calculates WER on test dataset
test: nemo_test.py
        @python nemo_test.py --test_path $(TEST_PATH) --model_name $(
           MODEL_NAME) --batch_size=$(BATCH_SIZE)

# Transcribes single or multiple audios in audio_directory
transcribe: nemo_test.py
        @python nemo_test.py --transcribe --audio_path $(AUDIO_PATH) --
           model_name $(MODEL_NAME) --batch_size=$(BATCH_SIZE)

# Transfers already pre-trained model to another language
transfer: nemo_train.py
        @python nemo_train.py --config-name $(CONFIG_NAME)
```

Listing A.4: Makefile created to ease the use of software

A.5 Dockerfile

```
# Fetch base image.
FROM nvcr.io/nvidia/pytorch:21.08-py3

# Get workdir from ARG so it can be changed when building.
ARG WORKDIR=/workspace

# Get NeMo from ARG so it can be changed when building.
ARG NEMO_VERSION=main

# Define workdir as an ENV to be available during run.
ENV WORKDIR=$WORKDIR

# Setting WORKIDIR to run the following commands.
WORKDIR $WORKDIR

RUN apt install -y libsndfile1
```

```
# RUN git clone -b $NEMO_VERSION https://github.com/NVIDIA/NeMo.git
# Setting WORKIDIR to install NeMo in a different location.
# WORKDIR $WORKDIR/NeMo
# RUN ./reinstall.sh
RUN pip3 install Cython
RUN pip3 install nemo_toolkit['all']==$NEMO_VERSION


# Update PyTorch version because of the GPUS being used.
RUN pip3 install torch==1.11.0+cu113 torchvision==0.12.0+cu113 torchaudio
   ==0.11.0+cu113 -f https://download.pytorch.org/whl/cu113/torch_stable.
   html
RUN pip3 install torchtext

# Install jupyter-lab.
RUN pip3 install jupyterlab

# Update the WORKDIR to be set for the launch.
WORKDIR $WORKDIR

# Exposes jupyter port.
EXPOSE 8888

# Setting up de run CMD to open bash.
CMD /bin/bash
```

Listing A.5: Dockerfile develop to create the environment where the developed software will run

UNIVERSIDADE DE ÉVORA
INSTITUTO DE INVESTIGAÇÃO
E FORMAÇÃO AVANÇADA

Contactos:
Universidade de Évora
Instituto de Investigação e Formação Avançada — IIFA
Palácio do Vimioso | Largo Marquês de Marialva, Apart. 94
7002 - 554 Évora | Portugal
Tel: (+351) 266 706 581
Fax: (+351) 266 744 677
email: iifa@uevora.pt