ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

DEPARTMENT OF COMPUTER SCIENCE
AND ENGINEERING

ARTIFICIAL INTELLIGENCE

MASTER THESIS

in

Text-To-Speech

Conditioning text-to-speech synthesis on dialect accent: a case study

CANDIDATE

Alessio Falai

SUPERVISOR

Prof. Zeynep Kiziltan

CO-SUPERVISOR

Dr. Federico Ruggeri

Academic Year 2020/21

Session 3rd

To my 1-hop neighbours.

ABSTRACT

Modern text-to-speech systems are modular in many different ways. In recent years, end-users gained the ability to control speech attributes such as degree of emotion, rhythm and timbre, along with other suprasegmental features. More ambitious objectives are related to modelling a combination of speakers and languages, e.g. to enable cross-speaker language transfer. Though, no prior work has been done on the more fine-grained analysis of regional accents. To fill this gap, in this thesis we present practical end-to-end solutions to synthesise speech while controlling within-country variations of the same language, and we do so for 6 different dialects of the British Isles. In particular, we first conduct an extensive study of the speaker verification field and tweak state-of-the-art embedding models to work with dialect accents. Then, we adapt standard acoustic models and voice conversion systems by conditioning them on dialect accent representations and finally compare our custom pipelines with a cutting-edge end-to-end architecture from the multi-lingual world. Results show that the adopted models are suitable and have enough capacity to accomplish the task of regional accent conversion. Indeed, we are able to produce speech closely resembling the selected speaker and dialect accent, where the most accurate synthesis is obtained via careful fine-tuning of the multi-lingual model to the multi-dialect case. Finally, we delineate limitations of our multi-stage approach and propose practical mitigations, to be explored in future work.

ACKNOWLEDGEMENTS

First, I would like to express my deepest appreciation to professor Kiziltan, for giving me the possibility to work on such an interesting and educational research project and supporting me throughout the entire experimentation and writing process. This extends to Federico, my co-supervisor, who additionally helped me feel relieved when certain research questions had to remain unanswered.

I'm also extremely thankful to all the amazing lecturers of the Artificial Intelligence course, who guided me in the discovery of many fields of study which I'm now very fond of. A special word has to be spent on professors Samuele Salti, Michele Lombardi and Paolo Torroni, for the uncountable *eureka* moments they sparked in me and for imparting true passion for their subjects.

I also feel very lucky to have met so many great educators throughout my educational path. Out of them all, I would like to acknowledge Grazia Biondi and Paolo Frasconi, who were the initiators of my curiosity in the two broad fields I enjoy the most: Computer Science and Artificial Intelligence.

Moreover, I will always be grateful to my study buddies, Leonardo, Lorenzo and Jacopo, with whom I have had the pleasure to tinker and experiment on Computer Science projects since high school. I must also thank my best friends, Saul and Flavio in particular, for their infallible support and sincere friendship.

My deepest gratitude obviously goes to my family, my parents and my brother, for teaching me the importance of hard work, for inheriting their ambitiousness and for providing me with all the resources and endearment necessary to achieve the long-term goals I set for myself.

Finally, the last word of appreciation goes to my life partner, Silvia, who has always been there to support and encourage me and act as a constant reminder of what's really important in life: love and happiness. Thank you for your patience, for your unconditional love and for standing by my side at all times.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ABI	Accents of the British Isles
AC	Accent Conversion
AD	Analogue to Digital
AI	Artificial Intelligence
ASR	Automatic Speech Recognition
BAP	Band Aperiodicity Parameter
BFC	Bark-Frequency Cepstrum
BFCC	Bark-Frequency Cepstral Coefficient
BN	Batch Normalization
CAPT	Computer-Assisted Pronunciation Training
CE	Cross Entropy
CER	Character Error Rate
CLIPS	Corpora e Lessici dell'Italiano Parlato e Scritto
CNN	Convolutional Neural Network
CSV	Comma-Separated Values
CV	Computer Vision

CVAE Conditional Variational Auto Encoder

DCF Detection Cost Function

DCT Discrete Cosine Transform

DET Detection Error Trade-off

DFT Discrete Fourier Transform

DL Deep Learning

DNN Deep Neural Network

DSP Digital Signal Processing

ECAPA Emphasized Channel Attention, Propagation and Aggregation

EER Equal Error Rate

FAC Foreign Accent Conversion

FAD Fréchet Audio Distance

FFT Fast Fourier Transform

FFTW Fastest Fourier Transform in the West

FID Fréchet Inception Distance

FRED FReiburg corpus of English Dialects

G2P Grapheme-to-Phoneme

GAN Generative Adversarial Network

GD Gradient Descent

GE2E Generalized End-To-End

GRU Gated Recurrent Unit

HMM Hidden Markov Model

HPC High-Performance Computing

IPA International Phonetic Alphabet

IViE Intonational Variation in English

kNN k-Nearest Neighbors

LSTM Long Short-Term Memory

MCD Mel-Cepstral Distortion

MFC Mel-Frequency Cepstrum

MFCC Mel-Frequency Cepstral Coefficient

ML Machine Learning

MLE Maximum Likelihood Estimation

MLP Multi-Layer Perceptron

MOS Mean Opinion Score

MSE Mean Squared Error

MT Machine Translation

MUSHRA MUltiple Stimuli with Hidden Reference and Anchor

NER Named Entity Recognition

NLP Natural Language Processing

OOV Out Of Vocabulary

PCM Pulse Code Modulation

PER Phoneme Error Rate

PESQ Perceptual Evaluation of Speech Quality

POS Part-Of-Speech

QA Quality Assurance

ReLU Rectified Linear Unit

RIR Room Impulse-Response

RNN Recurrent Neural Network

ROC Receiver Operating Characteristic

RP Received Pronunciation

S2S Sequence-to-Sequence

SCL Speaker Consistency Loss

SDR Signal to Distortion Ratio

SE Squeeze-and-Excitation

SGD Stochastic Gradient Descent

SOTA State Of The Art

SPSS Statistical Parametric Speech Synthesis

STFT Short-Time Fourier Transform

STT Speech-To-Text

SVD Singular Value Decomposition

SVM Support Vector Machine

t-SNE t-distributed Stochastic Neighbor Embedding

TDNN Time-Delay Neural Network

TTS Text-To-Speech

UMAP Uniform Manifold Approximation and Projection

V/UV Voiced/UnVoiced

VAD Voice Activity Detection

VAE Variational Auto Encoder

VC Voice Conversion

VITS Variational Inference with adversarial learning for end-to-end Text-to-Speech

W&B Weights & Biases

WAV Waveform Audio File Format

WER Word Error Rate

CHAPTER 1

INTRODUCTION

In this chapter, we aim to touch on the fundamental topics that will be extensively covered in this thesis, to give a high-level overview to the impatient reader. First, Section 1.1 provides the big picture on the history of Text-To-Speech (TTS) and what is current in the field; then Section 1.2 sets the research direction and summarizes the main contributions of this thesis. Next, Section 1.3 introduces related work and how this thesis differs from it. Finally, Section 1.4 outlines the document structure, by concisely explaining the content of each chapter.

1.1 TTS overview

Synthesising speech has been one of the ultimate goals of humankind since as far as 200 years ago when the vision was to build **mechanical apparatus** to artificially simulate the human vocal tract [160]. In the 20th-century, the advent of electronic devices enabled researchers to ditch mechanical solutions and transition to more powerful and flexible **digital representations**. The first digital TTS system was built based on the same reproduction of human articulators mentioned above [88], even though many more popular paradigms started to emerge. In particular, **concatenative systems** and unit selection solutions [72], in which small audio snippets are merged together, were all the rage in the $1950 - 1980$ period, until the recent Artificial Intelligence (AI) revolution came to be. Nowadays, thanks to the vast availability of computational resources, the latest and most

accurate TTS systems are all based on **neural architectures** [5, 185].

A modern TTS system is usually composed of **different modules**, that can either be trained jointly or at different stages [165, 166]. The first step is to convert raw characters to phonemes, which represent units of sound. Then, an acoustic model [145, 185] is tasked to predict an intermediate representation of speech starting from such phonemes and a final vocoder [100, 131] takes this representation and outputs the waveform that most closely matches the provided linguistic content. There are many alternatives to the reported pipeline, with more and more emphasis on **end-to-end solutions** [47], to go directly from character symbols to waveforms, to avoid the propagation of errors of multi-stage approaches.

One of the reasons to work with intermediate speech representations is that waveforms do not align well with the **one-to-many** nature of TTS systems, given that two similar waveforms could refer to very different linguistic contents. The opposite reasoning also holds, meaning that a single piece of text may correspond to multiple waveforms. This is because speech encodes much more than just linguistic content. For example, two different speakers are very likely to pronounce the same sentence in disparate ways, as both are influenced by their unique background, which includes age, location, social status, cognitive biases and much more.

The idea of modelling the so-called **prosodic properties** of speech, i.e. all the attributes remaining after removing the textual component, is what powers most of the contemporary research streams in the TTS field. To do so, TTS systems borrow techniques and architectures from many other disciplines, such as Automatic Speech Recognition (ASR). For example, models from the **speaker verification** [44, 93, 157, 181] literature, where the goal is to understand whether or not two utterances come from the speaker, are often employed to condition TTS synthesis on speaker identity, to alter the output and make it sound as if it was uttered by the selected speaker [32, 77].

The output of a TTS system, or the intermediate representation produced by its acoustic model, can also be used for downstream tasks. For example, Voice Conversion (VC) [104, 141, 142] is a speech-to-speech task that has the goal of converting **speaker identity**, while keeping linguistic content as is. In this case, a VC model could be used to convert the output of a mono-speaker system and simulate a multi-speaker one. VC systems have also been used for other purposes, such as data augmentation, to train TTS models on speakers for which not enough recordings are available [103].

All of the above contribute to the major objective of making TTS systems as **controllable** as possible, giving the user the possibility to tweak many different aspects of speech. Some examples, other than changing speaker identity, include adjusting the speaking rate (i.e. how fast or slow to utter) [141], making the model emote (e.g. sounding sad or happy) [78, 84, 106], sing [3, 25], laugh [173] and whisper [34]; and the list goes on with many more dimensions.

Recent advancements in Deep Learning (DL) and neural models for TTS have also given rise to more ambitious objectives, such as the ones related to **multi-lingual** [129, 200] and **polyglot** [126] models. In the former case, a single model is trained to speak many different languages with each language having its own associated speaker identity, while in the latter case a single model aims to maintain the same identity across multiple languages.

1.2 Motivations and contributions

All of the approaches presented in Section 1.1, such as the ones in the multi-lingual and polyglot streams, ignore more fine-grained modelling of language varieties within a country. In particular, previous work either focuses on completely different languages (such as English and French) or different locales of the same language (such as American English and British English).

We believe that shifting the attention towards **dialects and regional accents** would be beneficial in many ways. In particular, even though written languages are usually standardized in a country, their spoken counterparts may be subject to non-negligible variability that depends on many factors. Out of them all, location and education are usually attributed to be the largest contributors of such irregularities. In many countries, people who don't have access to higher education and those who live out of urban areas tend to communicate in ways that differ from what's regarded as the standard language. The gap between such standards and each variation of the language is very difficult to measure, but it's part of the cultural heritage of a country and should be preserved as such.

Thus, the benefits of providing accent control at the regional level for a speech system are multiple. One of them is to ease the approach of the uneducated user to modern technology and another one is to pass on knowledge of a country, as intimate as its linguistic

traditions and history. We also see a possible **business application** for voice assistants, such as Google Assistant and Amazon Alexa, which could take advantage of localization to more closely customize the user experience by relying on geographical location. In this way, someone living in Scotland would be able to hear a Scottish variation of the voice assistant's identity, while someone living in Ireland would hear a variety of British English closely resembling Irish English.

Given the above, the aim of this thesis is to give the user even more freedom, by adding a new *controllability dimension* to TTS systems, namely dialect accent. To the best of our knowledge, no prior work exists in the exploration of regional accents in a full TTS scenario. Thus, we summarize our contributions as follows.

First, we research for available **open-source corpora** containing information about dialect accents of the recorded speakers and decide to settle for what we will refer to as SLR83 [42], which is a dataset of more than 30 h of speech from 120 speakers spanning 6 broad **dialects of the British Isles**[1], i.e. Southern, Midlands, Northern, Irish, Scottish and Welsh.

Then, we explore how well-established and State Of The Art (SOTA) approaches in the speaker identification and speaker verification world might be adapted to their dialect accent counterparts. In particular, we compare a baseline d-vector [181] model and the cutting-edge TitaNet [93] model on the **dialect accent classification** task and perform detailed latent space analyses to check for speaker and accent leakage in dialect and speaker embeddings, respectively.

Next, we adapt a VC model, namely AutoVC [142], to the Accent Conversion (AC) task, where the goal is to **convert accent** while keeping both speaker identity and linguistic content fixed. To do so, we rely on our pre-trained speaker and dialect accent embedding models, to either convert speaker identity and dialect accent information jointly or just the latter.

On the TTS side of things, we mainly rely on the popular Tacotron 2 [152] architecture and propose a number of **practical pipelines** to convert text to speech uttered with the desired speaker identity and regional accent. We also compare our custom approaches with an end-to-end model from the multi-lingual field, namely YourTTS [20].

[1]The British Isles is a geographical term which refers to the two large islands that contain the mainlands of Scotland, Northern Ireland, the Irish Republic, Wales, and England, together with a large number of other, smaller islands that are part of the territories of these countries [96].

Finally, we conduct an extensive **experimental study** where we first verify the correctness of our custom implementations and assess the efficacy of individual components, to then evaluate the whole TTS pipelines from start to finish. In this regard, results show that each module achieves the intended goal when tested separately from the others, but fails to output satisfactory results when evaluated jointly. We attribute this shortcoming to the propagation of errors between the many modules that are part of the TTS pipeline and provide practical mitigations and directions for future research in the last chapter of the thesis.

It has to be noted that we are not proposing new architectures per se, but rather taking different **building blocks** that are nowadays mainstream to build TTS systems, and putting them together to show that such architectures have enough capacity to model regional variations of a given language (British English in our case).

As a further contribution, we **open-source** our code and evaluations through a publicly available GitHub repository[2] and we provide a collection of demo samples for most of the reported systems in a blog post[3]. Moreover, in case corrigenda and other resources will be needed in the future, they will be made available in a different GitHub repository[4], the same one holding the source LaTeX files for this document. We hope the linked resources will stand the test of time.

1.3 Related work

The controllable TTS stream of research introduced in Section 1.1 can be thought to be part of the more general flexible TTS one. One of the pillars of flexible TTS systems is **adaptive TTS**, where topics such as voice adaptation [23] and voice cloning [6] are key. All such systems share the common objective of adapting a source TTS model to a certain target, where the target could be a voice, a language, an accent, a style and more, using techniques such as few-shot and zero-shot learning[5]. Adaptive TTS also finds its way in many business applications, such as custom voice [22], a TTS service in commercial speech platforms that aims to adapt a source TTS model to synthesize personal voice for

[2] https://github.com/Wadaboa/tts-dialects

[3] https://alessiofalai.it/blog/master-thesis-samples

[4] https://github.com/Wadaboa/master-thesis

[5] For zero-shot adaptation (such as the one performed in [19]), a pre-trained encoder is mandatory, in order to extract embeddings for unseen entities, such as speakers not present in the training data; however, this scenario usually falls short in terms of adaptation quality and target similarity [165].

a target speaker using few speech data.

Furthermore, adaptive TTS comes in handy in **low-resource settings**, i.e. in those settings where target data is limited. Examples of low-resource settings include modelling minority languages (e.g. Vietnamese, Swahili, Hindi), exotic styles (e.g. newscaster, sarcastic) or simply speakers for which not enough high-quality recordings are available. In these cases, data from other languages/styles/speakers can be leveraged to improve the synthesis quality of the under-resourced target. For what regards cross-lingual transfer, one possible solution is to fine-tune pre-trained TTS models, where the pre-training happens with the high-resource languages, while fine-tuning is performed on the low-resource target language [175]. Other approaches (that also work in a zero-shot fashion) rely on shared grapheme/phoneme representations, such as International Phonetic Alphabet (IPA) [66] or byte-encoding [64], so that input text is mapped to a single latent space, regardless of the language it is written in. Dialect accents (or regional accents) may also be considered as low-resource languages, thus potentially allowing us to exploit prior work in such streams of research.

Other ways to achieve flexibility in TTS systems is by dealing with **disentanglement**, i.e. the concept of learning representations of speech attributes independent of each other. Possible ways to obtain such independent representations is by relying on well-established techniques, such as adversarial training and information bottlenecks [172]. Regarding the former, Wang et al. propose to rely on auxiliary losses to obtain speaker-independent text representations, by forcing a speaker classifier's output probabilities to obey a uniform distribution[6]. Regarding the latter, Qian et al. propose an extension of their popular AutoVC [142] architecture, so as to disentangle multiple speech properties (i.e. rhythm, pitch, content and timbre) using different bottleneck reconstructions. To the best of the author's knowledge, no prior work has been done in the disentanglement of regional accent (or even accent as a whole) from other speech properties such as linguistic content and speaker identity.

Also, approaches for disentanglement may rely on either explicit or implicit information, as described in [165]. Speech attributes for which labels can be extracted, either automatically or through a manual process, fall under the **explicit information** category, while everything else has to be considered as **implicit information**. Usually, attributes such as speaker/accent/language information are part of the former, while prosodic-level

[6]Intuitively, encouraging a classifier to follow a uniform distribution has the effect of confusing it, as the objective is to avoid it being able to distinguish the speaker classes.

features could be regarded as implicit. An example of how to deal with such implicit information is by using a so-called **reference encoder** [155], where prosody embeddings are extracted from reference audio at training time, while at test time a different reference might be used to reproduce its prosodic structure. An extension of the reference encoder [2] includes the use of a Variational Auto Encoder (VAE) to learn a latent space from which prosody embeddings can be sampled at inference time, thus allowing even more flexibility (e.g. exploring interpolations in the latent space to create new styles). Another extension of the reference encoder is with the popular **style tokens** [184], where prosody embeddings are used as part of an attention mechanism to attend to a bank of style tokens; then, at inference time the model can either rely on attention to select the best mix of style tokens, or such tokens can be forced by the user to allow some kind of style transfer to happen. In this thesis, disentanglement is always performed on the explicit side, as we rely on labelled data for both speaker and dialect accent information.

Regarding the **overall pipeline**, going from text to controllable speech, much inspiration is taken from [77], where a multi-speaker extension of the Tacotron 2 architecture is proposed. The key takeaway from [77] is that pre-trained speaker embedding modules can be used to condition the mel-spectrogram generation of an acoustic model. We rely on this result and extend this concept to the multi-accent case. Moving to the AC module, the closest work is described in [186], where a voice and accent joint conversion approach is proposed. The main difference with our solutions is that they adopt a recognition-synthesis framework, for which well-trained accent-dependent speech recognizers are mandatory to obtain meaningful bottleneck features. Moreover, they model 2 Chinese dialects, namely standard Mandarin and Tianjin-accented Mandarin, while our work focuses on 6 regional accents of the British Isles. Some parallels can also be made with [153], where authors propose to use style transfer techniques from the Computer Vision (CV) world for converting between British and American accents. Nevertheless, most of the regional accent investigations have been done in the ASR field, to solve challenges such as accent classification [41, 128] and speech recognition as a whole [149].

1.4 Document structure

Chapter 2 presents a broad introduction to the topics of Machine Learning (ML), DL, speech processing and TTS. Each section aims to be self-contained, meaning that someone who's expert in DL but has never worked with speech can directly jump to the last two sections in the chapter.

Chapter 3 starts with an overview of the original contributions of this thesis, to then dive deep into each component of the proposed TTS pipelines. For each module, we devise a section to introduce the task it was designed to solve and how it can be used as part of a TTS system. Then, we provide a thorough description of the main architectures, the adaptations needed to work with regional accents, and possible ways to evaluate each component.

Chapter 4 presents the extensive literature review for the selection of the SRL83 dataset and presents supporting data resources needed to verify the correctness of our custom implementations. We provide a common set of statistics for each dataset, related to the distributions of characters and the duration of utterances. Then, we outline all the pre-processing steps needed for training and testing each module of the TTS pipeline, for both text and audio data types.

Chapter 5 details the experimental study carried out to assess the correctness and efficacy of our proposed solutions and how they compare with SOTA architectures from the multi-lingual field. In particular, we first present the experimental setup and evaluation strategy common to all modules and then delineate steps and parameters specific for each model in their corresponding section. The structure of the experimental study mirrors the order in which models are described in Chapter 3. In particular, we first report results for individual components and then assemble them together to evaluate the whole TTS pipeline.

To conclude, in the last chapter we summarize the results observed in the experimental study and provide practical solutions to attenuate the propagation of errors of our multi-stage approach. Moreover, we devote the final bits of this thesis to set future research directions.

CHAPTER 2

BACKGROUND

In this chapter, Section 2.1 first provides the reader with a comprehensive introduction to the fields of ML and DL. Then, Section 2.2 introduces the main concepts behind speech signals and their representation on a digital machine. Finally, a wide overview of both legacy and modern TTS systems is presented in Section 2.3, along with their challenges and SOTA solutions. Each section in this chapter is self-contained, meaning that the reader can feel free to skip one of the sections if already familiar with the concepts therein.

2.1 Learning

In the most general sense, AI concerns the study and application of **rational agents**, i.e. agents that perceive and act in an environment, as thoroughly described by Russell and Norvig in their landmark book *Artificial Intelligence: A Modern Approach*. In a more narrow sense, this thesis only considers rationality as the ability to generalize, which intuitively means to perform sound inferences on previously unseen pieces of data. In the words of Russell and Norvig, the rational agent has to act well enough even on percepts it has never before experienced.

On a more practical scale, a rational agent might be considered as a function f that is completely defined by its set of beliefs (usually referred to as parameters) θ. The agent has the goal of learning a mapping between perceptions x (which we may as well name

inputs) and actions (which will be referred to as outputs), i.e. $y = f(x; \theta)$. To act well, the agent has to minimise the number of incorrect actions taken (or some other objective) and adjust its future actions accordingly, by shifting its beliefs. This iterative process, whereby the agent acts, receives feedback from the environment and updates its beliefs, is called **learning**. In case the agent is an artificial one, ML is the right term to use.

2.1.1 Supervised learning

The most common ML formulation is in the so-called **supervised** scenario. Let $D = \{(x_i, \hat{y}_i)\}_{i=1}^{N}$ be a set of N i.i.d. (independent and identically distributed) observations drawn from an unknown distribution, where X is the input domain and Y is the corresponding output (or label) domain. As described above for the rational agent, we also assume labels $\hat{y}$ to be generated by an unknown function f, s.t. $\hat{y}_i = f(x_i; \theta^*)$, fully parametrized by θ. At this point, the learning problem reduces to the estimation of the best-fitting parameters $\theta^* = \min_\theta \sum_{i=1}^{N} L(f(x_i; \theta), \hat{y}_i)$, where L is a loss function that measures how close predictions $y_i = f(x_i; \theta)$ are to ground truths $\hat{y}_i$.

Usually, the input domain X is assumed to be a Euclidean space $\mathbb{R}^d$ of large dimension d. In this sense, each observation $x_i = [x_{i,1}, \ldots, x_{i,d}]$ would actually be a vector in $\mathbb{R}^d$, meaning that each entry $x_{i,j}$ would describe a different feature (or peculiarity) of the input. For problems that are inherently easily described in a tabular format, adopting a **numerical representation** for the input data is straightforward. For example, if the objective is to predict whether or not an individual is likely to be wealthy (where wealthy is defined as earning more than 50 thousand dollars per year) based on census data[1], the set of features in Table 2.1 might suffice. In this example, each row is an observation $x_{i,*}$ and each column $x_{*,j}$ represents a different dimension that the learning procedure can exploit to minimize its loss function L.

As can be observed, only a subset of features in Table 2.1 is actually numerical, while some of them (i.e. *Education* and *Sex*) stem from what's called a **categorical distribution**[2]. Categorical variables need to be further processed before being fed to ML models, using techniques such as one-hot encoding, whereby column $x_{*,j}$ gets transformed into k binary columns (where k is the number of unique values attainable by feature j) s.t. if $x_{i,j} = v$, then $x_{i,v} = 1$ and $x_{i,u} = 0$ for all $u \neq v$.

[1] Examples and data are all based on the Adult dataset [49].

[2] A categorical distribution is a discrete probability distribution whose sample space is the set of k individually identified items [189].

Age	Education	Sex	Capital gain	Capital loss	Hours per week
39	Bachelors	Male	2174	0	40
50	Bachelors	Male	0	0	13
38	HS-grad	Male	0	0	40
58	7th-8th	Male	2936	0	40

Table 2.1: Subset of data from the Adult dataset [49]. Each row corresponds to a different individual from the census and each column represents a different attribute of such individual.

2.1.2 Towards representation learning

As input distributions become more complex, relying on standard ML techniques gets harder and harder. For example, how can images, words, speech and graphs be represented in a strictly numerical format? For decades, ML practitioners have relied on domain expertise and careful engineering of custom features to deal with such intricacies. Instead, nowadays more and more researchers are headed towards the field of **representation learning**, where objectives are twofold: models need to be able to perform well on downstream tasks (such as classification or regression), with the caveat of doing so via a learned representation. Given that we are working with numerical vectors, the model (based on its architecture) discovers such representation by finding lower-dimensional approximations and using non-linear higher-dimensional combinations of the original feature vector. The main motivation for higher-dimensional representations is that we can construct new features as non-linear combinations of the original features, which in turn may make the learning problem easier [40].

To stress the importance of representation learning, we first describe the **logistic regression** method, which is a probabilistic classification model, and then propose ways to address its limitations. In the simplest binary case, logistic regression aims to estimate the probability $P(y_i = 1 \mid x_i)$ that observation x_i is a member of the so-called positive class (which is class 1 in this example). Logistic regression solves this task by learning, from a training set D, a matrix of weights W and a bias term b: the weights W_j represent how important feature j is to the classification decision, and can be positive (providing evidence that the instance being classified belongs to the positive class) or negative (providing evidence that the instance being classified belongs to the negative class). Then, in order to force outputs $z_i = W \cdot x_i + b$ to be legal probabilities we pass

z_i through the sigmoid function[3] $\sigma(z_i) = \frac{1}{1+e^{-z_i}}$ (also called logistic function). After we have $P(y_i = 1 \mid x_i) = \sigma(z_i)$ we can threshold the output probability with t (the decision boundary) to obtain a discrete class, i.e. if $P(y_i = 1 \mid x_i) \geq t$ then we classify example x_i as belonging to class $y_i = 1$. In terms of the training procedure, logistic regression aims to minimize a Cross Entropy (CE) loss, which is a function that prefers the correct class labels of the training examples to be more likely, under the Maximum Likelihood Estimation (MLE) principle[4], and can be computed using Equation 2.1[5].

$$L_{CE}(y_i, \hat{y}_i) = -\log P(\hat{y}_i \mid x_i) = -[\hat{y}_i \log y_i + (1 - \hat{y}_i) \log(1 - y_i)] \tag{2.1}$$

One of the possible optimization strategies for logistic regression is **Gradient Descent (GD)**[6], which can be computed according to Equation 2.2.

$$\frac{\partial L_{CE}(y_i, \hat{y}_i)}{\partial W_{j,i}} = [\sigma(W_j \cdot x_i + b_i) - y_i] \cdot x_{i,j} = (y_i - \hat{y}_i) \cdot x_{i,j} \tag{2.2}$$

In case the number of examples in the training set becomes prohibitively large, the estimation of the gradient in Equation 2.2 may become computationally expensive. Hence, **Stochastic Gradient Descent (SGD)** is often employed, whereby outputs y_i and corresponding gradients are computed in batches (usually referred to as mini-batches) of B examples at once. Then, the mini-batch loss is the average of the losses for each example in the batch and the gradient is the average of the individual gradients. In this way, parameters are updated (at each training iteration) using an approximation of the true gradient, thus enabling convergence even when the number of training examples becomes very large [59].

One of the limitations of naïve logistic regression is that it is a linear model, i.e. it can only discriminate between linearly-separable classes (as the ones shown in Figure 2.1a), as z_i is just an affine transformation of the inputs x_i. To be able to linearly separate data points that are not linearly separable in input space, there's the need to map them to a different vector space, in which linear separability can be achieved. The way to do

[3] The sigmoid function is commonly used as it's differentiable in an elegant and easy way, i.e. $\sigma'(z_i) = \sigma(z_i)(1 - \sigma(z_i))$, it's nearly linear around 0 and tends to squash outlier values toward 0 or 1.

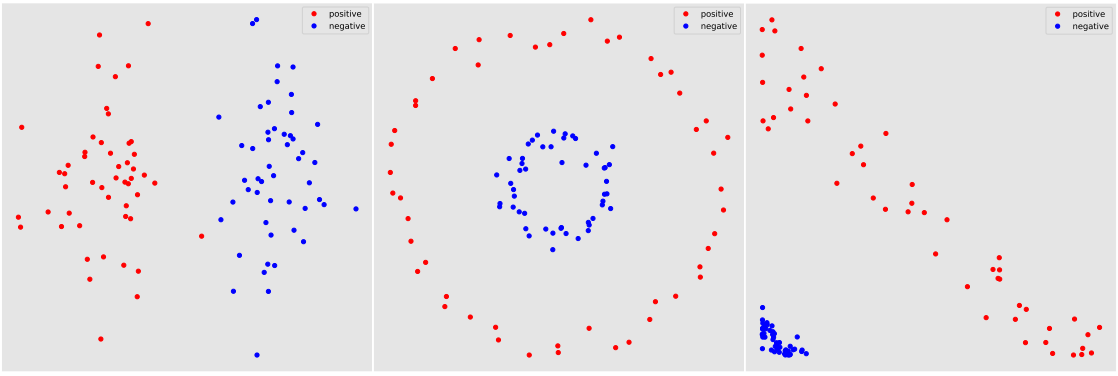
[4] Intuitively, MLE chooses the parameters that maximize the log-probability of the true labels in the training data, given the observations.

[5] The full derivation of CE is the following $P(\hat{y}_i \mid x_i) = y^{\hat{y}_i} \cdot (1 - y_i)^{1-\hat{y}_i} \equiv \log P(\hat{y}_i \mid x_i) = \log[y_i^{\hat{y}_i} \cdot (1 - y_i)^{1-\hat{y}_i}] = \hat{y}_i \log y + (1 - \hat{y}_i) \log(1 - y_i)$.

[6] For logistic regression, CE is conveniently convex, i.e. it has just one minimum, so GD starting from any point is guaranteed to find it.

so is by relying on **non-linear transformations**. For example, Figure 2.1 shows how mapping features $x_{i,1}$ and $x_{i,2}$, represented by the red and blue data-points in Figure 2.1b, to their squared counterparts can make the two classes linearly separable, as shown in Figure 2.1c.

Such non-linear transformations can either be handcrafted (as in the example above), in which case we talk about kernel methods that are usually employed in standard ML algorithms such as Support Vector Machines (SVMs); or learned through a stack of affine transformations interleaved by such non-linearities, in which case no domain expertise is required and we enter the realm of DL, neural networks, and representation learning. In this thesis, we'll mainly focus on the latter case, given that TTS systems rely on complex data such as text and speech signals.



(a) Inherently linearly separable data-points, i.e. we can draw a vertical line between red and blue dots.

(b) Inherently non-linearly separable data-points, represented by concentric circles in input space.

(c) Warping the input space, by squaring each dimension, to make data linearly-separable.

Figure 2.1: Examples of linearly (left) and non-linearly (middle) separable data and how to make the latter linearly separable in feature space (right). Red and blue data-points represent the positive and negative class, respectively.

Moving to the **neural network** scenario, we might consider logistic regression as a **Multi-Layer Perceptron (MLP)** with no hidden layers. Here, a layer is considered as an affine transformation followed by a non-linear one (usually referred to as **activation function**). As soon as one hidden layer[7] is added to the network's architecture, MLPs gain the so-called universal approximation property [37], meaning that they can approximate any continuous function of n real variables. Even though a single hidden layer suffices in the theoretical sense, empirical evidence suggests that adding more layers allows the network to learn better and better representations [162], where the term "better"

[7] A hidden layer is roughly defined as a layer between the input and output ones in a neural network.

is used to refer to semantically-meaningful properties.

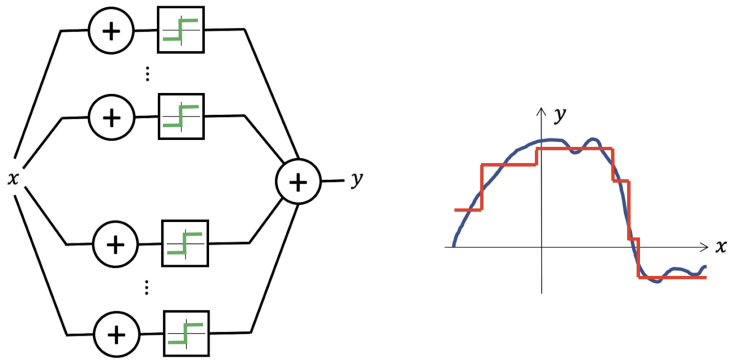


Figure 2.2: MLPs are universal approximators [14], i.e. the single-hidden-layer architecture on the left can model the smooth function on the right (blue curve) with a step-wise approximation of it (red curve).

2.1.3 About inductive biases

One of the core principles that has allowed the steady growth of DL is the use of **inductive biases**, which are ways to constrain the learning procedure based on prior knowledge. One way to inject inductive biases in neural networks is by restricting the set of functions they're allowed to learn. A popular example is with **Convolutional Neural Networks (CNNs)**, which exploit geometric priors such as shift (or translation) equivariance and invariance, to enforce the use of local features [14]. The former (shift-equivariance) is achieved through convolutional operators, while the latter (shift-invariance) can be obtained with layers such as pooling. The simplest examples in which CNNs excel and MLPs tend to fail is with image classification, where the classification result should not be affected by the position of the object in the image (thus exploiting shift-invariance), or image segmentation, where instead the output mask should shift along with the input image (thus relying on shift-equivariance).

Therefore, a convolutional layer holds a set of **shared weights** (instead of having a fully-connected matrix as MLPs do), also named kernels or filters, that encode the mentioned geometric priors. Assuming an input activation[8] A of shape $C_{in} \times W_{in} \times H_{in}$, a convolutional layer computes channel-wise convolutions between filters K (of shape $C_{out} \times C_{in} \times H_k \times W_k$) and the input, as shown in Equation 2.3 and depicted in Figure 2.3.

[8] An activation, or feature map, in a CNN could be the input signal (such as an RGB image) or the output of a convolutional layer.

Outputs for each filter are then aggregated (i.e. concatenated) along the output channel dimension to obtain a tensor of size $C_{out} \times H_{out} \times W_{out}$, where $H_{out} = \lfloor \frac{H_{in}-H_k+2P}{S} \rfloor + 1$, $W_{out} = \lfloor \frac{W_{in}-W_k+2P}{S} \rfloor + 1$, P is the amount of zero padding (on one side) and S is the stride (i.e. how fast we are moving in the input). In Equation 2.3, the $*$ symbol refers to the convolution operator, $K^{(j)}$ is the j-th kernel in the layer and b is a learnable bias term.

$$[K^{(j)} * A](i,j) = \sum_c \sum_m \sum_n K^{(j)}(c,m,n) \cdot A(c, i-m, j-n) + b \tag{2.3}$$

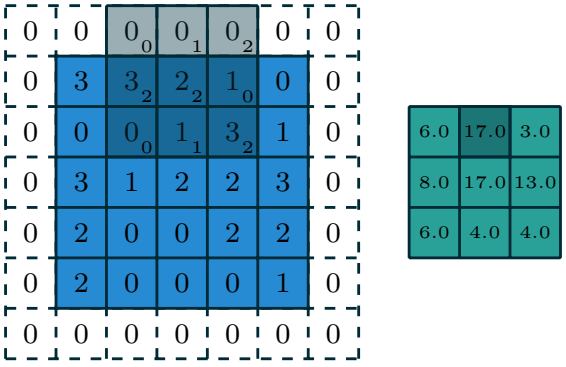
Different convolutional operators may also be applied depending on the task at hand. For example, **fractionally strided convolutions** (or transposed convolutions) are used in tasks such as image segmentation to learn to upsample[9] the input signal [147], while **dilated convolutions** are used in sequence to exponentially increase the CNN's receptive field[10] without coarsening the input feature map (which would be the case with simple strided convolutions) [197].

Another popular example of inductive bias is with **Recurrent Neural Networks (RNNs)**, where their vanilla versions are translation-equivariant, while their gated alternatives are also invariant to time-warping [14]. As introduced in [79], in a simple RNN (or **Elman network**) [51] the activation value of a hidden layer depends on the current input as well as the activation value of the hidden layer from the previous time step. The hidden layer from the previous time step provides a form of memory (or context), that encodes earlier processing and informs decisions to be made at later points in time. Hence, the main benefit of using RNNs over standard feed-forward networks is the ability to deal with inherently sequential inputs (such as natural language and speech) and the possibility to model sequences of varying length, which wouldn't be possible with plain MLPs.

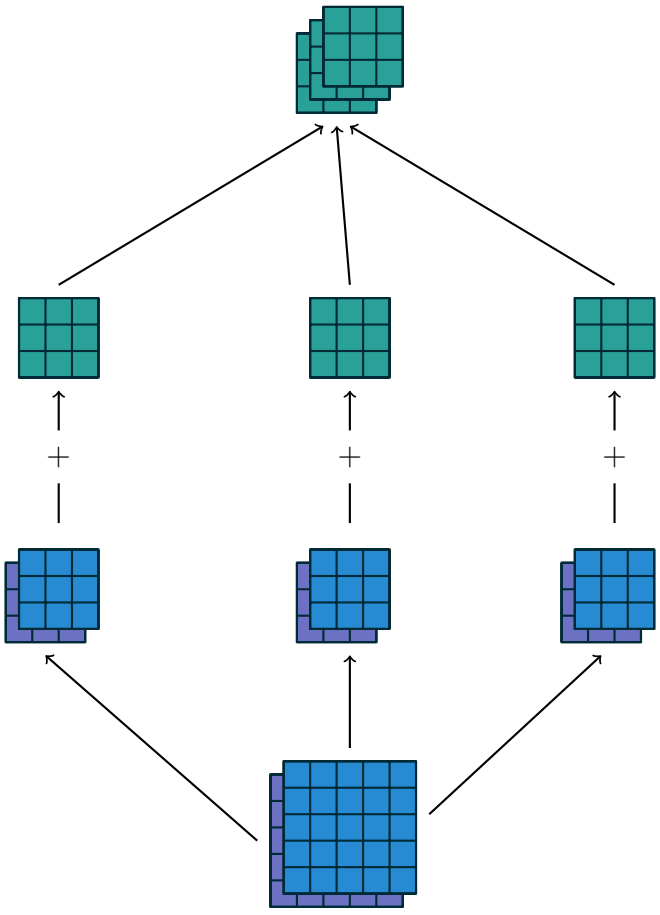
On a more formal note, in simple RNNs the hidden state at time t is computed as $h^{(t)} = g(Uh^{(t-1)} + Wx^{(t)})$, where parameters U, W are shared between each layer, g is a non-linear point-wise function such as $\tanh$ and $h^{(0)}$ is fixed. Given $x^{(1)}, \ldots, x^{(n)}$ inputs we obtain $h^{(1)}, \ldots, h^{(n)}$ hidden states (through time), that can be either directly mapped to an output (e.g. $y^{(t)} = f(Vh^{(t)})$, where f is usually softmax) or passed as input to

[9] Upsampling is a necessary part of segmentation pipelines, as most of them rely on encoder-decoder architectures, where the encoder maps images to semantically-rich, but coarse, feature maps, and the decoder has to exploit such information while also recovering the original input resolution.

[10] The receptive field in CNNs is the region of the input space that affects a particular unit of the network.



(a) Computing the output values of a discrete convolution.



(b) A convolution mapping from two input feature maps to three output feature maps.

Figure 2.3: Example of a convolutional layer [50], showing how to compute output values with a single kernel and input channel (above), along with a visualization of input-output shapes (below).

another RNN to obtain a stacked RNN and exploit the benefits of deep representations.

As explained in [164], ordinary RNNs are highly non-resilient to time rescaling, i.e. a task can be rendered impossible for the network simply by inserting a fixed, small number of zeros between all elements of the input sequence. Moreover, such networks suffer from vanishing and exploding gradient problems [9] when dealing with long input sequences, thus struggling to model long-term dependencies. These issues have led to the introduction of recurrent models with gating mechanisms, such as LSTM [69] and GRU [24]. Figure 2.4 shows an example of a **Long Short-Term Memory (LSTM)** network, which (in addition to what already explained for vanilla RNNs) relies on cell states $c^{(t)}$ that are preserved between computational steps. The cell state $c^{(t)}$ is updated based on what the forget gate $f^{(t)}$ lets through from the previous cell state $c^{(t-1)}$ and how much of the cell content $\tilde{c}^{(t)}$ (an intermediate cell state) the input gate $i^{(t)}$ selects. The hidden state is then the result of picking entries from the cell state $c^{(t)}$, based on the output gate $o^{(t)}$. **Gated Recurrent Units (GRUs)** are simply a more lightweight variant of LSTMs, as they combine the forget and input gates into a single update gate, while also merging the cell state and hidden state.

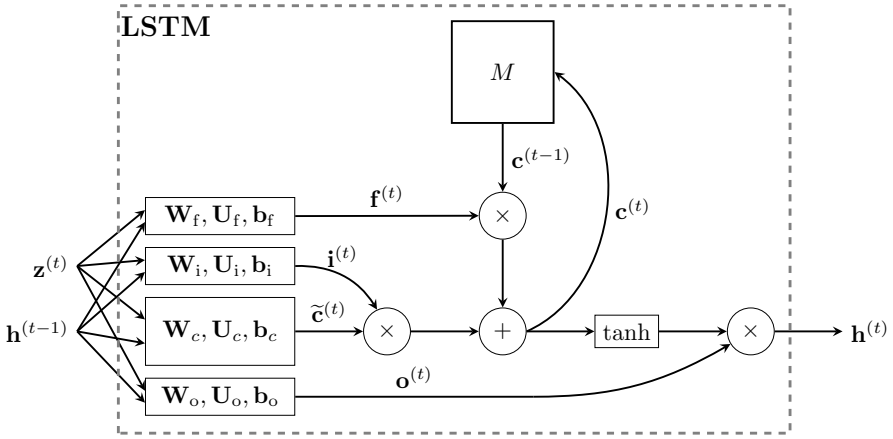


Figure 2.4: Flow of data in an LSTM layer [14]. Here, z is the input, h is the hidden state, c the cell state, f, i, o are the forget, input and output gates, M is the memory of cell states and other symbols represent learnable weights.

As with CNNs, extensions of RNNs have also been devised. One example is about **bi-directionality**, whereby the input is processed in two ways, from left to right (the *forward* pass) and from right to left (the *backward* pass); then, outputs of the two computations are merged (usually by concatenating them over their last dimension). The intuition behind bi-directional RNNs is that they tend to understand the overall context better than their

uni-directional counterparts since they can aggregate information from the *past* and from the *future*, which might be a useful property for tasks in the natural language domain, where meaning does not usually simply flow from left to right.

2.2 Speech

In this section, we are introducing speech signals, both from a linguistic and digital perspective. Subsection 2.2.1 starts with a broad explanation of communication and the elements of its spoken component, such as prosody, to then detail the concepts of dialects and accents. Then, Subsection 2.2.2 deals with how speech is produced by the human vocal organs and identifies important notions that will be carried over to the digital world, such as the use of fundamental frequency. Finally, Subsection 2.2.3 introduces the main useful Digital Signal Processing (DSP) concepts, such as ways to represent speech in a digital form and how to further process it to extract information in the way humans perceive sound.

2.2.1 Communication

Human communication can be thought of as having 2 principal dimensions, the **verbal** and the **prosodic** one [166]. The former relies on a symbolic system and deals with arranging words in sequences to form sentences, while the latter is used to emphasize meaning and help the listener better understand the underlying message.

Communication can also travel through different means, such as text or speech. In the former case, the concept of written communication is devised, whereby a message is structured in sentences made of words, which are in turn composed of **graphemes**[11]. Instead, spoken communication (at least its verbal component) has its principal unit of form in the **phoneme**. In the same way as text, phonemes can be mixed to form words and sets of words are then lined up to form meaningful sentences. Differently from graphemes, phonemes tend to vary less from language to language.

[11] In alphabetic languages like English and French, graphemes can be thought of as letters or characters, while in syllabic writing like Japanese, each grapheme represents a syllable, and in languages like Chinese each grapheme represents a full word or part of a word.

Prosody

The second component in human communication is **prosody** and it's the main dimension that discriminates spoken from written communication. Prosody can serve as a way to express emotions (e.g. anger, sadness, happiness), to indicate where a sentence ends and more, and it does so with stress, rhythm, intonation, tone and other phonological structures.

Prosody is fundamental in many applications, such as smart assistants, that cannot complement speech with other cues, like visual ones (e.g. hand gestures, head nodding), which is instead an important part of human communication, as it adds to the overall meaning conveyed by the speaker.

As already hinted above, the written signal contains little or no prosodic information and smart use of the verbal component has to be made to compensate for the lack of prosody. For example, if the goal is to convey sadness, a writer may write *"I'm over the moon. I finally got promoted to assistant regional manager."* instead of *"I'm doing fine. Today I got a promotion."* when answering the question *"How are you doing today?"*. Writers may also rely on emoticons to further stress specific aspects of their message: 😐 might mean that the writer is disgusted or not feeling very well, while 😄 means that the writer is somehow having fun or laughing at their correspondent's previous messages. Another way to embed prosody in written messages is to use punctuation and underlined italicised or bold text.

Taylor differentiates between 2 main forms of prosody, termed affective and augmentative. The **affective** use of prosody is probably the most common, as it deals with expressing primary and secondary emotions[12], speech acts (e.g. question) and modes of expression (e.g. sarcasm). Instead, the **augmentative** use of prosody serves as an addition to the verbal component in situations where its syntax can lead to confusion, as it relies on word emphasis (to focus the listener's attention on a specific concept) and phrasing structure (e.g. pauses) to nudge the listener's interpretation towards the one intended by the speaker.

[12]The term primary emotions refers to emotions which are supposed to be innate, while secondary emotions are assumed to arise from higher cognitive processes, based on the ability to evaluate preferences over outcomes and expectations [38].

Accents and dialects

As the main goal of this thesis is to include the dialect accent dimension as part of the speech properties that can be controlled in an end-to-end modern TTS system, we devote the next few paragraphs to explain what we mean by accent and introduce the notion of dialect from a linguistics perspective.

An **accent** can be regarded as a set of speech features (such as rhythm, stress and intonation) that are common within individuals belonging to a certain group. In other words, people who share the same location, ethnicity, social class or native language are more likely to also share the same accent.

Some common accent variations are labelled with adjectives. For example, the variation of accent between regions of the same country is referred to as **regional accent**. Another example is the variation of accent that is due to a speaker's native language, which is often recognized as **foreign accent**. Anyways, the definition of accent mostly comprises pronunciation-related attributes and, depending on which kind of accent variation one wishes to study, different features may be affected.

A **dialect** is instead considered as a superset of accents, meaning that to correctly discriminate between different dialects, more than just pronunciation-related features need to be considered, including other linguistic dimensions such as lexicon and grammar. The usual way of dealing with dialects is by clustering them in dialect areas, i.e. by arranging them along geographical lines, as done with the broader concept of languages, even though other factors (such as social class or ethnicity) may play an important role in the definition of dialect, as it happens for accents.

In this work, we mainly focus on variations of accents that are due to geographical reasons, meaning that we use the terms *accent*, *dialect accent* and *regional accent* interchangeably. In particular, we focus on the dialect accents of the British Isles.

Regarding accents of the **British Isles**, Received Pronunciation (RP) is usually presented as a supra-regional accent and it's the variation of British English most frequently taught in schools worldwide. RP is, still today, the main accent used in formal spoken communication (such as broadcast television) and is often referred to as the Queen's English or BBC English. Other than RP, the British Isles span many different regional accents, and care needs to be taken in their identification, as accents and dialects blend subtly and imperceptibly into one another, i.e. there is no way to mark regional cut-off points for ways of speaking [96]. Because of this, what we focus on in this thesis is the

exhibition of a set of features that most closely conform to a characteristic local way of speaking.

2.2.2 Phonetics

Phonetics deals with the study of **speech production** and **speech perception**. The former examines the processes by which humans convert linguistic messages into speech, while the latter aims at understanding the opposite process. Speech production is part of articulatory phonetics, while speech perception is part of acoustic phonetics. For the purpose of this thesis, we will only focus on speech production.

Something to keep in mind is that in the field of phonetics, phonemes are realised as **phones**[13] and if a phoneme has more than one phone the set of phones are called **allophones**. Moreover, the term **segment** is used as a general term to refer to phonemes, phones and allophones.

Speech production

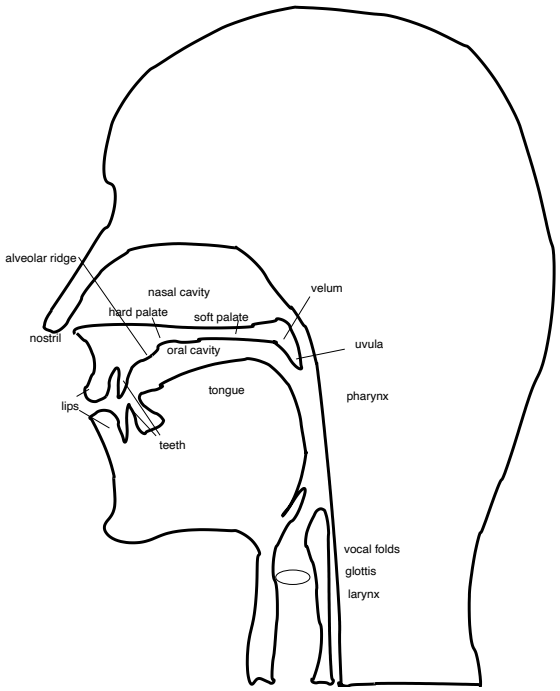


Figure 2.5: Diagram of the human vocal organs or articulators [166]. Sound is produced by tension in the vocal folds and modified by other articulators on its way out.

[13] Phones are "absolute" and are not specific to any language, while phonemes can be discussed only in reference to specific languages [189].

A speaker can use a combination of different vocal organs to produce a wide range of sounds. The main source of sound is produced by tension in the vocal cords (also known as vocal folds) and the degree of the opening between them (the gap between vocal folds is the glottis). A narrow opening causes vocal folds to vibrate, giving rise to a periodic sound identified by a so-called **fundamental frequency** or **F0**. Opening the glottis slightly further forces periodic vibrations to stop and generates what's called an **unvoiced** sound (as opposed to the **voiced** one produced with a narrow opening), which is the basis for certain speech processes such as whispering.

In voiced sounds, the speaker can vary the fundamental frequency of the produced sound, by increasing or decreasing the tension applied to vocal folds. As a consequence, **harmonics** will also change, as they represent energy sources at frequencies that are multiples of the fundamental. Harmonics give sound its basic **timbre**, while the fundamental is related to sound **pitch**.

Sound is thus generated by a basic source, but it can be modified on its way out, and other vocal organs, shown in Figure 2.5, come into play at this stage of the speech production process. For example, the vocal tract (which comprises the pharynx, the oral cavity and the nasal cavity) functions by altering the basic timbre, i.e. it tweaks harmonics' relative strengths, but it keeps the fundamental fixed. As soon as harmonics are amplified or attenuated in amplitude by the vocal tract, they are referred to as **formants**[14] and named **F1, F2, F3** and so on[15]. This model of speech, whereby source sound is generated and further altered by the positions of body parts such as tongue, jaw and lips, is named **source/filter model**. At the very last step of such a model, sound needs to be outputted: this can happen via the mouth (oral sounds), the nose (nasal sounds) or both (nasalised sounds).

One of the most difficult aspects of speech production is **co-articulation**: to a certain extent, two neighbouring phonemes have a joint articulation and this is because the articulators are constantly moving and the pattern of movement for a particular phoneme is heavily dependent on the phonemes preceding and following. This phenomenon has been studied for a long time in speech processing and linguistics and it's one of the main sources of complexity in building artificial models of articulators.

[14] An amplification caused by a filter is known as resonance in signal processing and formant is the corresponding term in speech processing.

[15] The fundamental frequency is named F0 but it's not a formant, as it is not modified by the vocal tract.

2.2.3 Signal processing

In this section, we transition from the linguistic type of notions introduced so far to talk about signal processing concepts. In particular, we first focus on the raw representation of a speech signal on a digital machine, to then explain how to further process such representations to obtain higher-level features, that will be useful for downstream tasks.

Waveform

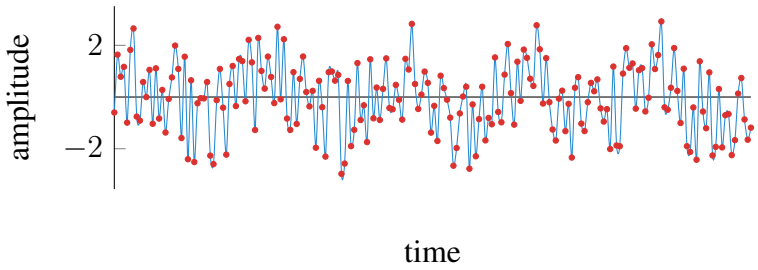


Figure 2.6: Example of an audio waveform. The blue line indicates the original analogue signal and the red markers are the selected sampling points.

All speech signals in the real world are **continuous signals** which describe the pattern of air pressure variation over time. These signals can be recorded with a variety of analogue means, but for computer analysis, we require our signals to be digitised such that the continuous signal is converted to a discrete one. This **Analogue to Digital (AD)** process is composed of 2 main tasks: sampling and quantization. The former depends on the **sampling rate** (or frequency) parameter, which intuitively indicates how many equispaced data points to extract over a specific time frame, while the latter aims to limit the (otherwise possibly infinite) range of amplitude values to a finite set. For example, a continuous signal might be sampled with a frequency of $8\,\text{kHz} = 8000\,\text{Hz}$, meaning that once every $\frac{1}{8000} = 0.000125\,\text{s}$ an amplitude value is extracted from the signal. After sampling, we end up with a sequence of amplitudes $x_n, n \in N$ that must be **quantized** to be represented in a digital system. This can happen by truncation or rounding or any other quantization method, by choosing a fixed number of bits to represent each datapoint with: a common choice is to rely on 16 bit integers, but there are also other options, such as 32 or 64 bit floats[16].

[16]There is usually a trade-off that needs to happen when selecting sampling rates and quantization bits in speech systems, as the higher they are, the better the data quality, the worse the memory efficiency and the higher the time required to process them.

The usual way to graphically represent a speech signal is with a **waveform**, i.e. by plotting time over amplitudes. Figure 2.6 shows an example waveform, in which the blue plot indicates the original analogue signal and the red markers are the selected sampling points.

Fourier transform

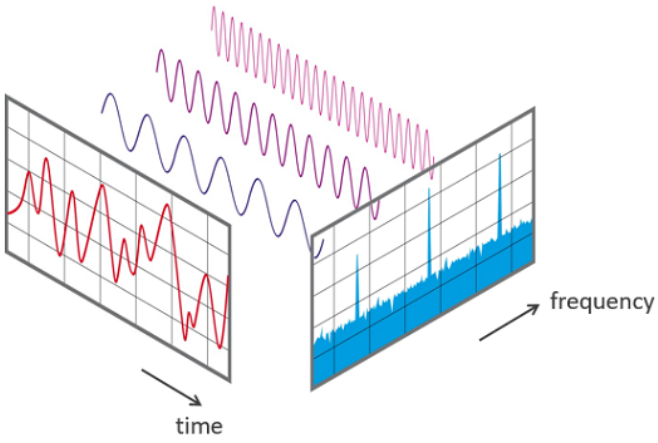


Figure 2.7: Fourier transform visualization (image courtesy of NTi Audio). The red and blue plots indicate the variation of amplitude over time and frequency, respectively.

The **Fourier theorem** states that any periodic function which is reasonably continuous may be expressed as the sum of sine and cosine terms (called the Fourier series). The **Fourier transform** is a way to achieve such decomposition and it can be applied to both continuous and discrete signals. When applied to discrete signals it is referred to as Discrete Fourier Transform (DFT) and it's usually expressed with Equation 2.4.

$$y_k = \sum_{n=0}^{N-1} x_n e^{-\phi i} \overset{(2.5)}{=} \sum_{n=0}^{N-1} x_n \cos(\phi) - x_n \sin(\phi) i, \text{ with } \phi = 2\pi \frac{kn}{N} \tag{2.4}$$

In Equation 2.4, x is our discrete signal, x_n is the n-th element of x, N refers to the number of observations, i is the imaginary unit, y_k are the main components of the discrete transform (the amplitudes of the waves that are employed to reconstruct the original signal) and $e^{-\phi i}$ is the frequency of the component. In order to transition from polar to Cartesian coordinates in the complex plane, we can rely on Euler's formula and split $e^{-\phi i}$ into cosine and sine terms, as in Equation 2.5.

$$e^{\phi i} = \cos(\phi) + \sin(\phi) i \tag{2.5}$$

Plotting amplitudes over frequencies gives the so-called **spectrum** of the signal, which intuitively tells us which frequencies play more important roles than others. As shown in Figure 2.7, the spectrum is represented in the frequency domain (frequency on the x-axis), while the original signal is in the time domain (time on the x-axis) and both have amplitudes on the y-axis. Using the inverse DFT formula we can also go the other way around, from frequency to time domain, as reported in Equation 2.6.

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} y_k e^{\phi i} \tag{2.6}$$

The **phase spectrum**, that encodes in which way signals are delayed for each frequency component, can also be computed through Fourier transform. One important consideration is that the human ear is largely insensitive to phase information in discerning speech and other sounds. Thus, two waveforms with the same magnitude spectra will sound the same regardless of phase, which is instead only used to localise sounds. This is why in speech analysis only the magnitude spectrum is taken into consideration. Nevertheless, the phase spectrum is essential for speech synthesis, as it is needed to correctly reconstruct the original waveform when starting from the frequency domain.

Fourier transform, in its discrete variant, is quite popular also thanks to its **computational efficiency**. As mentioned above, Fourier transform decomposes the input signal into a number of waves, each one with an amplitude and a frequency. Once the number of observations N is known, frequencies are also known (they are fixed), but amplitudes y_k need to be tuned so as to make Equation 2.6 hold: for the discrete case this turns out to be relatively simple, as there is a closed-form formula which tells exactly what the values of each y_k should be. The worst-case time complexity to compute a DFT decomposition is $O(N^2)$, but there are faster algorithms, termed **Fast Fourier Transform (FFT)** methods, that are able to achieve an upper bound of $O(N \log N)$[17].

Fourier analysis has lots of interesting applications, spanning from seasonality detection in time-series data to speeding up convolutional operators (thanks to the equivalence between convolution in the time domain and multiplication in the frequency domain). For what regards speech, the spectrum can be used to derive representations that in some way better encode the information which we are interested in than simple waveforms, as we shall see in the following sections.

[17] One of the fastest implementations of DFT can be found in the Fastest Fourier Transform in the West (FFTW) C package, available at the following address: `https://www.fftw.org/`.

Windowing

One of the problems of applying DFT over speech signals is that they are usually **non-stationary**, meaning that their statistical properties change with time. Thus, performing standard DFT on discretized audio snippets would result in a spectrum that does not convey useful information, due to the "random" frequency components that might be observed. We can get around this problem by assuming that the speech signal is in fact stationary if considered over a sufficiently short period of time (this is because long windows are not able to capture transitions in spectral content). Therefore we model a complete speech waveform as a series of short term frames of speech, each of which we consider as a stationary time-invariant system.

This approach, termed **Short-Time Fourier Transform (STFT)**, is specified by multiple steps. First, a **windowing function**, along with its parameters, has to be selected to extract frames of speech (common windowing functions are depicted in Figure 2.8). Then, frames of speech s_n are obtained from the full waveform x_n by multiplication by the chosen window w_n in the time domain: $s_n = w_n \cdot x_n$. Finally, DFT is applied to each windowed segment to obtain a sequence of short term spectra. Other important aspects to keep in mind are related to the size of each window and the selected stride, or **hop length**, which represents the amount of shift between consecutive windows.

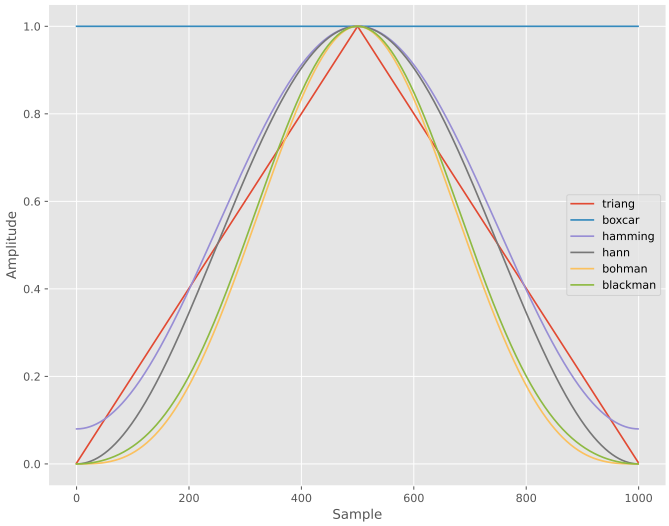


Figure 2.8: Common STFT windowing functions. Windows are used to obtain a local view on the signal and compute DFT over amplitude values inside the window.

The usual way to represent the computed sequence of short term spectra is using a so-called **spectrogram**, which is a 3-dimensional plot having time and frequency on the

horizontal and vertical axis, respectively, and amplitudes as colour or brightness intensities. Figure 2.9 shows an example of such representation.

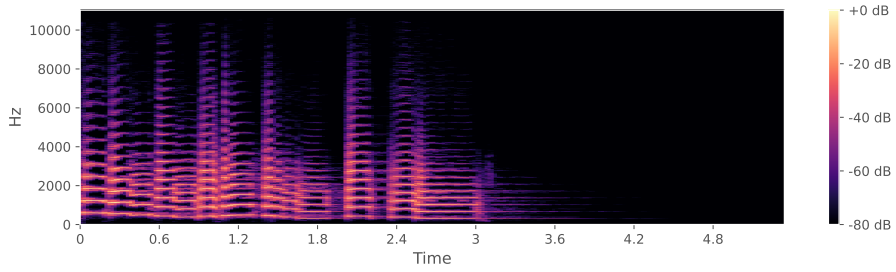


Figure 2.9: Example of a spectrogram. Each pixel in the image represents the decibel value of the analysed signal at a specific time and frequency.

In order to reproduce human sensitivity to sound, spectrograms are usually represented in **mel-scale** and the logarithm operation is computed over the resulting mel-spectrogram. The mel-scale is used because humans are more sensitive to lower frequencies, while the logarithm is taken because humans are less sensitive to slight differences in amplitude at high amplitudes than at low ones (a doubling in sound only produces an additive increase in perceived loudness). The mel-scale intuition can be implemented in practice by leveraging non-uniform window sizes and strides, so as to be more discriminative at lower frequencies and less discriminative at higher frequencies. Figure 2.10 shows an example of a mel filter bank that uses a triangular windowing function, while formulae in Equation 2.7 are used to transition between Hertz (f) and mel (m) units of measure.

$$\begin{aligned} m &= 2595 \log_{10}(1 + f/700) \\ f &= 700(10^{m/2595} - 1) \end{aligned} \tag{2.7}$$

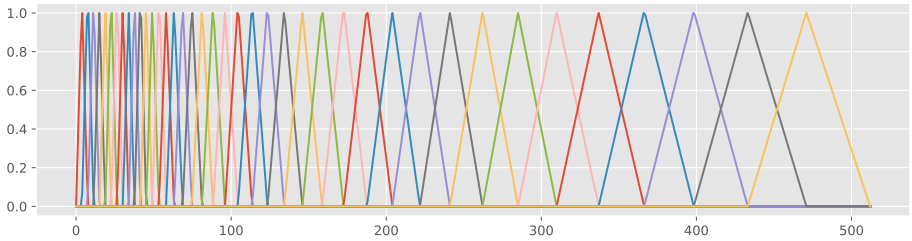


Figure 2.10: Example of a triangular mel filter bank. Note how windows at higher frequencies are more spaced out w.r.t. the ones at lower frequencies.

Acoustic features

As spectra and waveforms are high-dimensional and highly correlated from one window to the next one, low-dimensional approximations of them might also be leveraged for some applications. The **cepstrum** is a well-known example of this. Starting from a spectrum, the cepstrum is obtained by computing the Discrete Cosine Transform (DCT) of the log-amplitudes in the spectrum, as if it were a signal. As stated in [31], there are 8 different versions of DCT, with type 2 being the go-to reference. DCT type 2 can be computed using the formula in Equation 2.8 [13].

$$y_k = 2 \sum_{n=0}^{N-1} x_n \cos\left(\frac{\pi k(2n+1)}{2N}\right) \tag{2.8}$$

The DCT is highly similar to the DFT: whereas in a DFT the basis functions are sinusoids, in a DCT they are restricted solely to **cosines**. A signal's DCT representation tends to have more of its energy concentrated in a smaller number of coefficients when compared to the DFT, and is thus commonly used for signal compression [13]. In case DCT is computed on a mel-scaled spectrum, the cepstrum is termed Mel-Frequency Cepstrum (MFC) and its amplitudes are referred to as Mel-Frequency Cepstral Coefficients (MFCCs). The cepstrum can also be computed on more exotic scales, such as the **Bark** one, a psychoacoustical[18] scale (similar to the mel one) that defines 24 barks based on the first 24 critical bands of hearing. To convert Hertz (f) into Bark (b), and vice-versa, we can rely on the formulae reported in Equation 2.9 [174].

$$\begin{aligned} b &= \frac{26.81f}{1960+f} - 0.53 \\ f &= \frac{1960(b+0.53)}{26.28-b} \end{aligned} \tag{2.9}$$

In case DCT is computed on a Bark-scaled spectrum, the cepstrum is termed Bark-Frequency Cepstrum (BFC) and its amplitudes are referred to as Bark-Frequency Cepstral Coefficients (BFCCs).

Another family of acoustic features is related to speech **aperiodicities**, as they are essential for high-quality waveform synthesis. As reported in [154], a binary voicing decision, referred to as Voiced/UnVoiced (V/UV), describes whether the signal is voiced

[18] Psychoacoustics is the branch of psychophysics involving the scientific study of sound perception and audiology, i.e. how humans perceive various sounds [189].

or not i.e. whether there is a fundamental frequency (F0) associated with the signal or not. However, even for the speech segments defined voiced the vocal-cord vibration is not perfectly periodic. The amount of devoicing, occurring especially in the high frequencies, is described by the voice aperiodicity, where aperiodicity is defined as the power ratio between the speech signal and the aperiodic component[19] of the signal [124]. Usually, aperiodicities are computed as averages in a pre-defined number of frequency bands, such as $[0-1, 1-2, 2-4, 4-6, 6-8]$ kHz, giving rise to the Band Aperiodicity Parameter (BAP) feature.

2.3 TTS

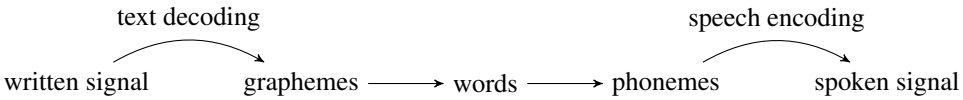


Figure 2.11: The common-form model [166], which shows the flow of a signal from written to spoken form, through the successive encoding of it in graphemes, words and phonemes.

TTS, or **speech synthesis**, is the process of generating speech from some kind of textual input representation, as opposed to ASR systems that follow the opposite flow (from audio to text). Nowadays TTS and ASR methods are ubiquitous, having the greatest accomplishments in smart assistants (such as Amazon Alexa, Google Assistant, Apple Siri and Microsoft Cortana), call-centre automation and many more business applications.

This section provides an overview of the main topics and areas of research in speech synthesis, starting from its key constituents and the most successful historical models, all the way up to SOTA architectures and training procedures.

2.3.1 Legacy systems

TTS systems became popular for the diverse range of applications they have. One of the first real use of this technology was in reading systems for the blind, in which a synthetic voice is tasked to read books and articles out loud and help the disabled person to navigate around some kind of ad-hoc operating system.

[19] A speech signal may be separated into a periodic and an aperiodic component using specialized algorithms, such as the ones described in [124, 196].

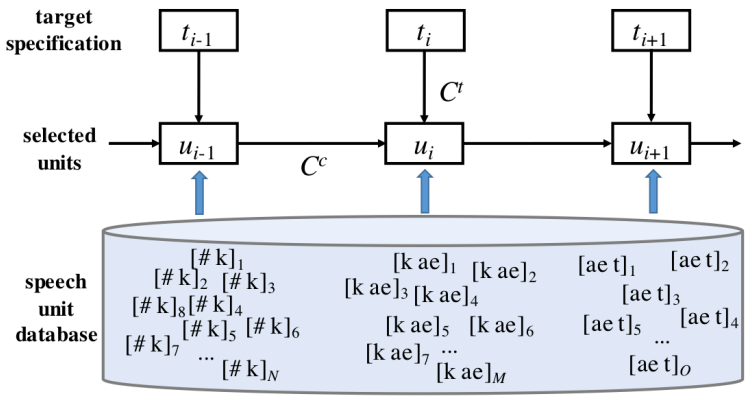


Figure 2.12: Example of cost to synthesize the word *cat [k æ t]* in a unit selection system [176].

Before the advent of neural architectures, a diverse set of techniques were employed to make the synthetic voice sound as intelligible and natural as possible, even though the quality levels that could be reached were quite limited and far behind human performance.

The most common legacy techniques can be clustered in 3 categories: articulatory, formant and concatenative synthesis. We can position the different synthesis methods along a *knowledge about speech* scale, as done in [15]: articulatory synthesis needs a considerable understanding of the speech act itself, while concatenative systems tend to be on the opposite side and formant models are positioned in the middle of the scale, as they generally look for *higher-level parameters* or to larger pre-stored units than concatenative models.

Articulatory synthesis is related to the creation of physical models of human articulators and its popularity gradually declined because of the scarcity of data that would be required for proper articulator simulation.

Formant synthesis, also referred to as rule-based synthesis, requires a considerable amount of work by domain experts, as they need to work out the formant structure and other spectral properties of speech as closely as possible. Speech is then synthesized by various signal processing modules that are dependent on a range of parameters, such as the formant values, bandwidth, voicing, F0 and more.

On the other side, **concatenative synthesis** abstracts from human characteristics and relies on copy-pasting pre-recorded samples of audio to obtain intelligible speech. The main problem with concatenative approaches is that in their naïve versions they produce choppy results, with discontinuities at snippet boundaries, resulting in less natural voices.

Best results are obtained when working with **sub-word units**, such as diphones and triphones[20], to solve co-articulatory issues and reduce the number of recordings needed.

Early work in concatenative synthesis relied on **diphone synthesis**, in which professional speakers recorded one sample for each diphone in the target language and signal processing techniques were applied on the concatenated signal to ensure naturalness. Subsequent solutions rely on **unit selection**, in which speakers record entire sentences (thus resulting in many copies of the same diphone) and correct diphones (or more generally, units) are extracted by cost minimization using two cost functions: the target cost $C^t(u_i, t_i)$ and concatenation cost $C^c(u_{i-1}, u_i)$. An example of such costs is shown in Figure 2.12. As reported in [176], the target cost describes the mismatch between the target speech unit specification t_i and a candidate unit u_i from the database, while the concatenation cost describes the mismatch of the join between the candidate unit u_i and the preceding unit u_{i-1}. Unit selection greatly improves diphone synthesis thanks to its ability to capture both local and global effects (e.g. word-level and syllable-level structures), even though speech quality is not on par with Statistical Parametric Speech Synthesis (SPSS) and neural methods.

For some applications, legacy techniques might still suffice. In particular, sci-fi movies commonly rely on old TTS methods to enrich the dramatic component of certain scenes by deliberately making the synthetic voice sound more *robotic* or *computerised*: this is because current TTS technologies generate sound that's similar or on par with the one produced by human articulators and the movie audience is more used to associate a robotic-sounding voice, rather than a more natural sound, to a non-human being [79]. Also, intermediate forms of automated speech production, such as **phrase splicing** [48] (also referred to as canned speech or pre-recorded prompts), might be very useful for applications like announcements on transportation systems (such as trains and buses) or speaking clocks. In the former example, the train staff records common sentences like the one reported in Figure 2.13 (without the content in angular brackets), along with all the possible train names (e.g. *Frecciarossa*), train numbers (e.g. *9956*), train stations (e.g. *Roma Termini*) and hours of the day (e.g. *21.30*); then, the phrase splicing system fills the gaps in the common sentences with the appropriate audio snippets or using speech synthesis for novel content (e.g. a new train station).

[20] A diphone is an adjacent pair of phones in an utterance, a triphone consists of triples instead of pairs, and so on. As n grows, more and more context is incorporated and intricacies such as co-articulation become less of an issue as the transition between phones is included in the unit itself.

"Welcome aboard the light speed train <TRAIN-NAME> <TRAIN-NUMBER>. The train will stop in <STOP-LIST> and will arrive in <DESTINATION> at <TIME>."

Figure 2.13: Example of a train announcement with phrase splicing. The content in angular brackets is what needs to be recorded separately from the template sentence.

2.3.2 Modern components

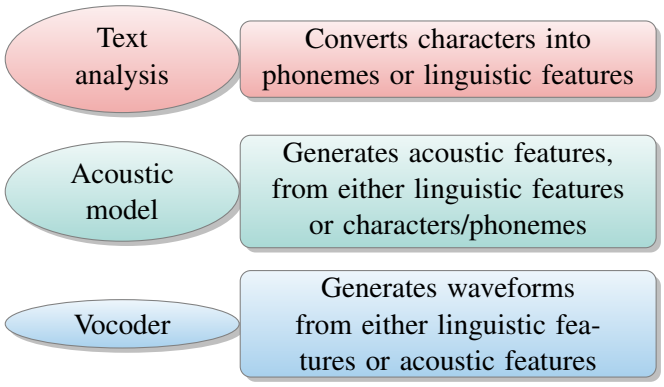


Figure 2.14: Main TTS components. Raw text is taken as input and further processed by the frontend module. Then, the acoustic model produces an intermediate speech representation, which is converted to a raw waveform by the vocoder.

Figure 2.14 shows the 3 main components that are part of all modern TTS systems, as identified in [165]. The usual pipeline starts from raw textual input, which could be a sequence of words, characters or, more generally, graphemes. The **text analysis** module (also referred to as frontend) converts the input representation into a set of linguistic features, such as Part-Of-Speech (POS) tags and suprasegmental information, or phonemes. The output generated by text analysis is then used as input for the subsequent **acoustic model**, which takes care of building acoustic features such as spectrograms. Finally, acoustic features are given to a **vocoder**, that is tasked to generate the correct waveform from the intermediate representation.

There are some notable exceptions to the standard pipeline depicted in Figure 2.14: researchers have tried to skip specific steps of the workflow, giving rise to a family of different models. For example, models such as GAN-TTS [12], EATS [47] and EfficientTTS [123] are headed towards the end-to-end direction, i.e. they generate the final waveform output from either characters or phonemes. Such end-to-end processing has several practical advantages: first, it reduces time-to-market thanks to the simplified maintenance required by a single model vs multiple ones; and second, it gives the entire

model the ability to be optimized on a single metric, the one we care about the most for downstream tasks.

Text analysis

As described in [166], the job of the text analysis system is to take arbitrary text as input and convert this to a form more suitable to subsequent linguistic processing.

There are multiple steps in the frontend module and they all add up to the overall complexity of the system. One of the first tasks that should be tackled is **text normalization**, which comprises 2 sub-tasks. The first is **tokenization**, which aims to extract tokens from raw text and its output might depend on the language we are working with. For example, word boundaries are generally more difficult to be identified in character-based languages such as Chinese. The second sub-task is text normalization is **verbalization**, a Sequence-to-Sequence (S2S) problem that takes as input a word or set of words targeted for text representation and outputs a normalized variant of them that's more suited for synthesis. Common examples are dates, times, acronyms, currencies and more. Complications for this task are given by the one-to-many relationship between input and output representations. For example, the time *10:09 am* can be translated as *"nine past ten in the morning"* or *"ten oh nine a m"* or simply *"ten oh nine"* and the correctness of the output might depend on the context. This step can be carried out by rule-based systems, function approximators or a combination of the two.

The next stage of the text analysis process is **homograph resolution**, in which raw text is augmented with linguistic features to remove ambiguity and aid subsequent models achieve better speech synthesis. The ambiguity stems from words having the same form, but different meanings. For example, the word *"address"* can be used in contexts *"The business had moved to a new address"* and *"We need to address the problem of absenteeism"*, but in the first sentence its meaning is associated to that of a location, while in the second phrase it means "to give attention to a problem". In this example, the two uses of the word *"address"* require different pronunciation, meaning that it is a homograph but not a homophone. For the purpose of TTS, homographs that are not homophones have the greatest importance, as they entail that different sounds need to be produced out of the same word form. To solve this issue, we can rely on well-established techniques for POS tagging, for which a POS label is attached to each input token. In our example, the first use of *"address"* would be associated with a `NN` label (noun), while the second one

to a VB label (verb), thus resolving ambiguity. In certain cases POS tagging alone might not be sufficient to discriminate homographs. For example, in the sentence *"I have an apple"* the word *"apple"* could mean the fruit or the company and POS tagging would assign the same NN label (noun) in both cases. In these situations, other Natural Language Processing (NLP) techniques such as Named Entity Recognition (NER) can come in handy, as they could be able to tag the *"apple"* word with the ORG label (organization), in case the user is referring to the company, and this would all be based on surrounding tokens[21]. Both POS and NER can be implemented as dense classification models and nowadays SOTA results are achieved with sequence-based neural networks, such as RNNs and Transformers.

Next, **prosody prediction** aims to predict a prosodic form for each linguistic and acoustic segment from the text. One of the most important aspects of prosody prediction is end-of-utterance detection, which deals with sentence splitting and is one of the phenomena that listeners are most sensitive to [166]. If an utterance ends with a question or exclamation mark, then the task is relatively simple, but most of the ambiguities are related to periods, as they can be associated with both end-of-abbreviation and end-of-utterance. For example, in the sentence *"Specifically w.r.t. the unemployment problem"* the word *"w.r.t."* is an abbreviation for *"with respect to"* and periods inside *"w.r.t."* may confuse the sentence splitter module. End-of-utterance detection is usually solved with relatively simple ML methods, such as decision trees, because of their interpretability and the possibility to inject custom rules in them [133]. Other than sentence splitting, prosody prediction deals with identifying variations in syllable duration, loudness and pitch to predict rhythm, stress and intonation of speech and all such procedures are based on tagging systems to label each kind of prosody [165]. It's also very common to compute prosodic forms for different text granularities (such as phoneme, syllable, word, phrase and utterance level) and concatenate them in a single feature vector.

The final step is **Grapheme-to-Phoneme (G2P) conversion**, i.e. generating a sequence of phonemes from a sequence of characters. This step is mostly solved with rule-based techniques, even though care should be taken to make sure Out Of Vocabulary (OOV) words are correctly translated.

[21] This step is only required for homographs that are not homophones. In the "apple" example, the word most probably has the same pronunciation for both the fruit and the company, but it's still reported to stress the potential impact of a NER model in such situations.

Acoustic model

The acoustic model is tasked to convert graphemes/phonemes or linguistic features to acoustic ones. Solutions that output acoustic features such as F0, MFCC, BFCC and BAP fall under the umbrella of SPSS and rely on either Hidden Markov Models (HMMs) or Deep Neural Networks (DNNs) to learn proper mappings between linguistic and acoustic features. One of the main issues in SPSS-based modelling is that of alignment, meaning that there usually isn't a one-to-one correspondence between textual features and speech properties. Instead, solutions that compute high-dimensional spectrograms (either linear or mel-scaled), in place of low-dimensional features such as those based on cepstrum, tend to perform much better because they require less pre-processing (they usually directly take characters/graphemes or phonemes as input) and they provide an easy way to learn alignments (through attention or explicit duration prediction) [165].

One of the first acoustic models to be composed entirely of neural building blocks is **DeepVoice** [5], shown in Figure 2.15.

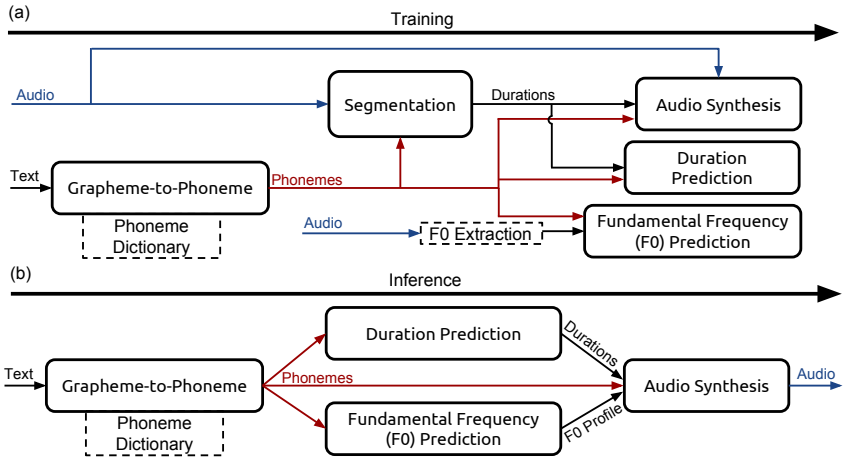


Figure 2.15: DeepVoice architecture [5], showing the speech production process at training (above) and inference time (below).

DeepVoice comprises the following five main modules.

1. A G2P module to convert English characters to phonemes, which are deterministically mapped using the CMU dictionary[22] or using a S2S model in the case of OOV words;

[22] http://www.speech.cs.cmu.edu/cgi-bin/cmudict

2. A segmentation model, that identifies phoneme start and end indices in an audio file, which is only used at training time as a data labelling approach to provide ground truths to the phoneme duration model;

3. A phoneme duration prediction model, that predicts the temporal duration of every phoneme in a phoneme sequence (an utterance);

4. A fundamental frequency prediction model, which predicts F0 throughout the phoneme's duration (only if the phoneme is voiced);

5. An audio synthesis model, that combines the outputs of previous modules to synthesise audio at a high sampling rate.

Unlike prior work, DeepVoice only relies on phonemes with stress annotations, phoneme duration, and fundamental frequency (F0) as acoustic features, which are then fed to a vocoder to generate raw waveforms.

Further work has been carried out by the same authors with DeepVoice 2 [7] and DeepVoice 3 [139], with the former being a multi-speaker extension of the first iteration, and the latter being a fully-convolutional variant that inherits some of the good findings from related work, such as the use of mel-scaled log magnitude spectrograms as acoustic features.

Vocoder

The vocoder (a contraction of voice and encoder) is the final module in the TTS pipeline and it's used to generate waveforms from either linguistic or acoustic features. It was originally introduced to synthesise speech in SPSS systems, with well-known implementations such as STRAIGHT [80] and WORLD [125], and further gained popularity with neural models such as WaveNet [131]. Vocoders used in SPSS are bound to take as input acoustic features such as F0, MFCC and BAP, since they are the main output of SPSS-based acoustic models. Instead, neural vocoders tend to work with magnitude spectrograms and are based on the concept of **phase reconstruction**, to recover the missing phase spectrum and obtain higher quality waveforms. The phase spectrum is mandatory because an arbitrary signal cannot be reconstructed by means of an inverse Fourier transform when only its magnitude is known. Other non-neural algorithms can also be used with spectrogram-based vocoders, with the most popular one being Griffin-Lim [62].

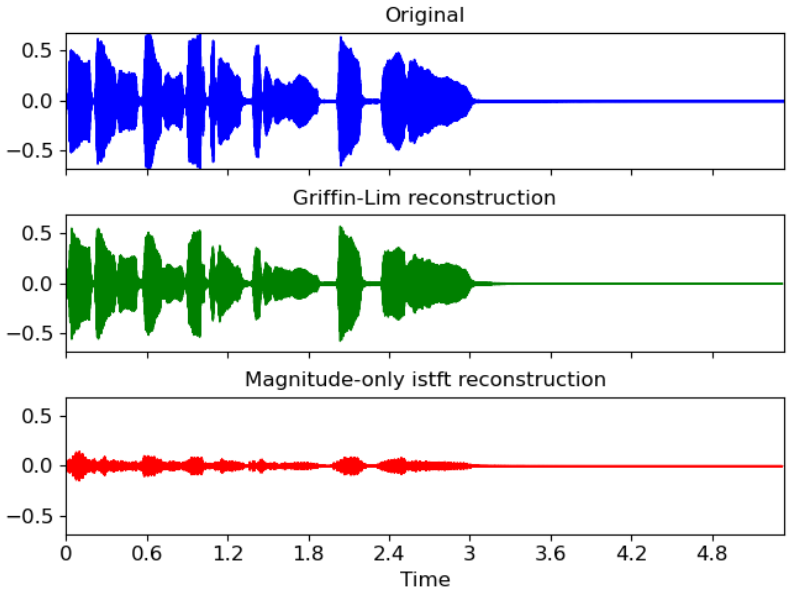


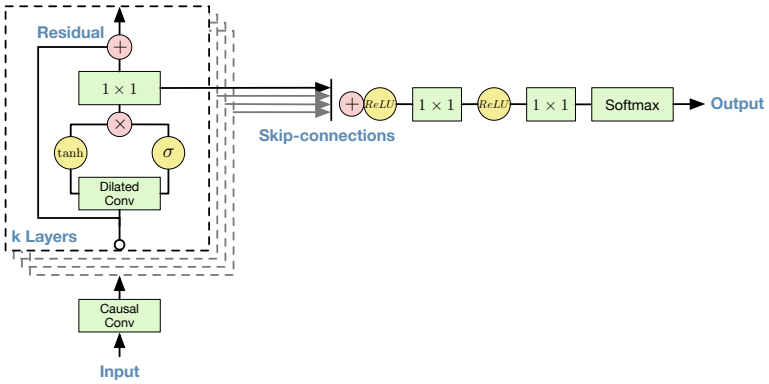
Figure 2.16: Griffin-Lim vs naïve inverse STFT [119]. Notice how the Griffin-Lim reconstructed waveform (green) is way closer to the original signal (blue), than the one obtained with an inverse STFT without phase information (red).

As described in [163], the **Griffin-Lim** method [62] is a signal-processing-based iterative algorithm to reconstruct a phase spectrogram from the amplitude spectrogram. Let $y = [y_1, \ldots, y_T]$ and $z = [z_1, \ldots, z_T]$ be amplitude and phase spectrograms, respectively. Vectors $y_t = [y_{t,0,}, \ldots, y_{t,f}, \ldots, y_{t,F}]$ and $z_t = [z_{t,0,}, \ldots, z_{t,f}, \ldots, z_{t,F}]$ represent the amplitude and phase values at frame t, respectively, f is the frequency index and F corresponds to the Nyquist frequency. Both $y_{t,f}$ and $z_{t,f}$ are real-valued variables, but only $z_{t,f}$ is a variable with a period of 2π. The Griffin-Lim method randomly initializes z first. Then, it iteratively takes an inverse STFT to obtain a waveform from y and z, followed by an STFT to re-obtain z from the waveform. In the loop, y is substituted for the newly computed one and the process goes on until convergence or a maximum number of iterations is reached. The method can reconstruct the phase spectrogram consistent with the given amplitude spectrogram, but makes some artefacts in the synthesized speech, such as extra reverberation and phasiness owing to inappropriate initialization of z. Figure 2.16 shows the difference between a magnitude-only inverse STFT and the waveform that can be obtained with the more accurate Griffin-Lim algorithm.

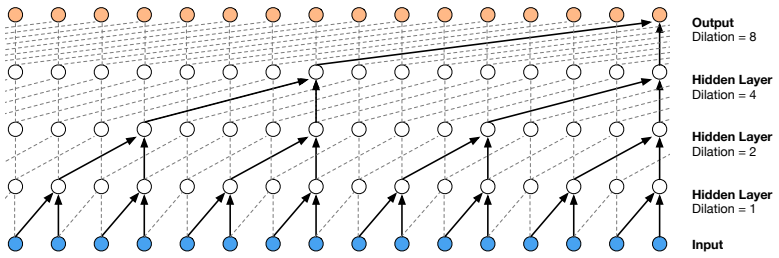
There are also notable **variations of the standard Griffin-Lim** method, such as its *fast* variant [138], that introduces a momentum parameter α to give more control over phase estimates updates. If α is correctly tuned and $\alpha \in [0, 1]$, it can lead to faster convergence, but whenever $\alpha > 1$ original convergence properties are lost. Griffin-Lim can

also be implemented in its *deep* variant [118], which stacks a sub-block including two Griffin-Lim-inspired fixed layers and a DNN for incorporating prior knowledge of target signals into phase reconstruction. In this work, authors formulate Griffin-Lim as a parameter-fixed RNN consisting of STFT and inverse STFT layers within the network, along with a trainable DNN made of convolutions, batch normalization, residual connections and gated units.

Recent advances in neural models also enabled the exploitation of much more powerful modelling techniques. In particular, **neural vocoders** are generative neural networks that can produce speech samples by sampling from a learned distribution, and WaveNet [131] was the first model of this kind.



(a) Architecture and residual blocks. The residual blocks are followed by 1×1 convolutions to control the number of activation maps and the output is a distribution over 16-bit amplitude values.



(b) Stack of dilated causal convolutions. The former property is used to exponentially increase the receptive field, while the latter one to respect the auto-regressive nature of the model.

Figure 2.17: The WaveNet model [131]. The model's architecture (above) comprises a stack of residual layers with dilated causal convolutions (below).

Inspired by the PixelCNN architecture [132], **WaveNet** relies on dilated causal convolutions, as shown in Figure 2.17b. The **causality** comes from conditioning the prediction only on previous time-steps, which can be thought of as performing a sequence of masked convolutions. Instead, the **dilation** is given by how the convolutional filters are

expanded while maintaining the same number of parameters. One of the big advantages of dilated convolutions is that stacking multiple layers with exponentially increasing dilation rate allows the receptive field to grow exponentially, while the number of parameters only increases linearly and input resolution is kept the same, as briefly introduced in Subsection 2.1.3.

The WaveNet model is **auto-regressive**, i.e. generating the speech sample at time t depends on all previous samples before t. This is both an advantage, since training can be parallelized (all time-steps of the ground truth are known a priori), and a disadvantage, since predictions need to be generated sequentially.

Regarding the actual architecture, shown in Figure 2.17a, WaveNet relies on a **stack of residual blocks** with gated activation units before the skip connection itself. Gates are the same as the ones used in PixelCNN and they were empirically observed to be better than standard Rectified Linear Unit (ReLU) units for modeling audio signals. Finally, the **output layer** is implemented with a softmax that computes a categorical distribution of size $65\,536$, because of the 16-bit integer values used to store waveforms. In practice, to ease the training procedure authors rely on μ-law quantization, so that the training data is transformed according to μ-law encoding, as in Equation 2.10.

$$\hat{x_t} = sign(x_t)\frac{\ln(1+\mu|x_t|)}{\ln(1+\mu)}, x_t \in (-1,1) \tag{2.10}$$

Instead, WaveNet waveforms are transformed with μ-law expansion, as in Equation 2.11.

$$x_t = sign(\hat{x_t})\frac{(1+\mu)^{|\hat{x_t}|}-1}{\mu}, \hat{x_t} \in (-1,1) \tag{2.11}$$

In this way, the final softmax layer need only output $\mu + 1 = 256$ values.

For TTS applications, WaveNet is also conditioned on the logarithmic fundamental frequency values in addition to linguistic features and phone durations. One of the strengths of WaveNet is that it can also be conditioned on further inputs. For example, in a multi-speaker setting the speaker identity can be used to select different speaking styles.

CHAPTER 3

CONDITIONING TTS ON DIALECT ACCENT

As already introduced in Chapter 1, a long-standing goal of TTS systems has been to gain the ability to learn representations of speech attributes independent of each other[1]. Fulfilling this objective would enable end-users to tweak individual speech properties, without affecting the others, thus opening a world of possibilities and business applications. Following this objective, in the following sections we thoroughly describe how this thesis extends previous research, providing detailed explanations of the overall work and each of its constituents.

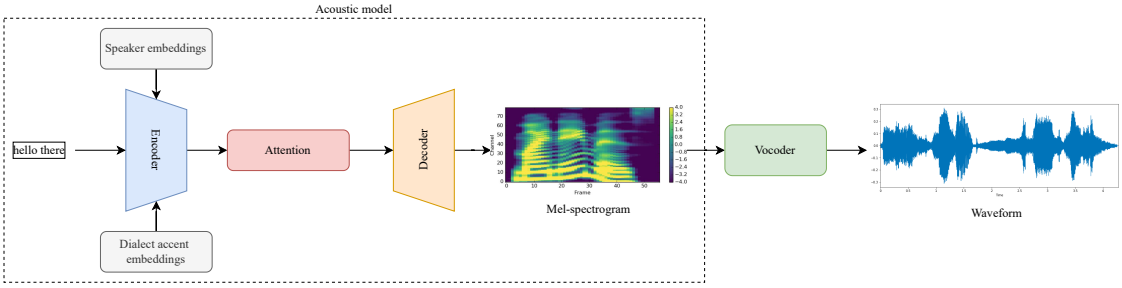
In particular, this thesis proposes 2 different pipelines to deal with the dialect accent dimension, as shown in Figure 3.1. All architectures are composed of 4 main components, i.e. embedding modules for both speaker identity and dialect accent information, which will be introduced in Section 3.1; an acoustic model to convert text to an intermediate speech representation, which is described in Section 3.2; a voice/accent conversion system to transform speech in the target speaker/dialect accent (see Section 3.3); and a final vocoder that converts the speech representation into a proper waveform (refer to Section 3.4).

The first pipeline, depicted in Figure 3.1a extends the work proposed in [77] to a multi-accent scenario. Instead, the second pipeline, reported in Figure 3.1b, relies on a

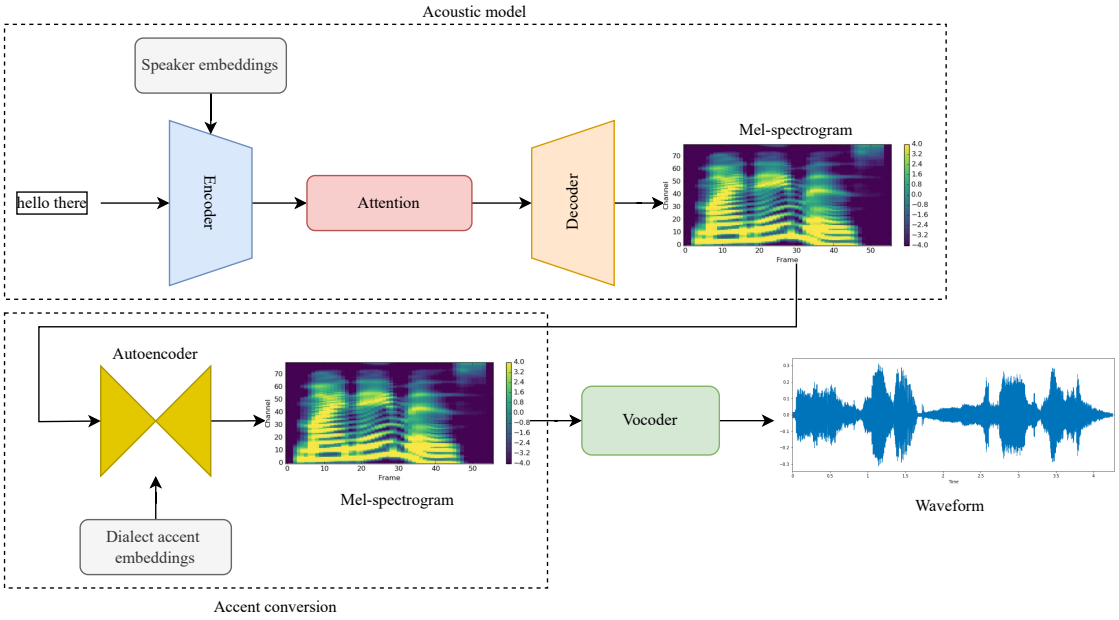
[1] We later refer to this core concept of independent representations as disentanglement.

standard multi-speaker acoustic model and achieves accent control through a novel AC system.

As a further contribution, we compare our proposed approaches with a cutting-edge SOTA multi-lingual model that performs joint multi-speaker and cross-lingual synthesis in an end-to-end fashion. In Section 3.5 we describe such model and the adaptations needed to work with regional accents.



(a) Acoustic model with speaker and accent conditioning.



(b) Acoustic model with speaker conditioning and accent conversion system.

Figure 3.1: Proposed architectures to control for dialect accent in TTS synthesis.

3.1 Speaker and dialect accent embeddings

Embedding modules are used to compress a high-dimensional signal in a low(er)-dimensional vector while keeping the main properties of the original input. Embeddings are projected on a so-called latent (or hidden) space, which is a manifold[2] that allows to model complex similarity relationships between input features. Such similarities are usually controlled through the training procedure of the embedding model, using custom loss functions that encode the semantics to propagate over to the embedding space.

In this thesis, embedding spaces have to follow two different objectives: the first one is to cluster together audio belonging to the same speaker (this is the **speaker embedding model**), while the second one is to form clusters of audio uttered by speakers in the same dialect (this is the **dialect accent embedding model**). In the following sections, we first introduce learning tasks and objectives to be used in audio embedding models, then describe the main metrics used to evaluate such models, and finally explain the baseline model used in subsequent experiments and the SOTA model taken from the speaker verification literature, adapted to work with both speaker and dialect classes.

3.1.1 Tasks and terminology

Audio embedding modules are usually trained on an identification task and evaluated on a verification one. For example, in speaker embedding models the training task is speaker identification and the evaluation task is speaker verification. Both speaker identification and verification are sub-tasks of the broader **speaker recognition** challenge. In the former, the goal is to understand from which of the registered speakers a given utterance comes, while the latter aims to accept or reject the identity claim of a speaker. Speaker recognition systems can also be divided in **text-dependent** and **text-independent**, with the former requiring the speaker to issue a pre-determined utterance, whereas the latter does not rely on a specific text being spoken. The main practical difference between text-dependent and text-independent systems is that the first one is usually easier to train and more reliable, while the second one requires much more data to account for the variability of a single speaker's speech.

Speaker recognition can also be categorised into **closed-set** or **open-set** settings [29]. In the former, all identities are predefined in the training set, thus no zero-shot abilities

[2] In mathematics, a manifold is a topological space that locally resembles Euclidean space near each point [189].

are required and a simple classification model might suffice. In the latter, some identities are not seen during training: this can be cast as a metric learning problem in which voices must be mapped to a discriminative embedding space.

To perform speaker identification using a speaker embedding model, input speech is mapped to the embedding space and compared to speaker centroids (extracted from the training set) to assess which one is closest in the latent space: the label associated with the closest centroid can then be interpreted as the speaker identified in input speech. There are multiple ways to **measure closeness**, but the most popular one is to use a k-Nearest Neighbors (kNN) approach based on some sort of distance metric, such as Euclidean or Manhattan, or similarity measure, such as cosine similarity. Instead, in speaker verification, the input is a pair of utterance and speaker label $\hat{l}$ and the goal is to assess whether or not the speaker identified by $\hat{l}$ is the one speaking in the given utterance. In order to do so, speaker identification is performed on the input utterance, producing speaker label l, and labels are compared as $l == \hat{l}$.

The same tasks described for speakers can be generalized to dialects, thus spanning the **dialect recognition** task and its constituents' dialect identification and dialect verification sub-tasks. The case of full dialect recognition is strongly simplified over dialect accent recognition, as the former can rely on dialect-specific lexicons to discriminate different dialects, while the latter has to rely on speech alone and models need to be able to disentangle speech identity from dialect accent information.

In this thesis, we'll mainly focus on text-independent and closed-set speaker/dialect recognition systems. Also, variations of dialects due to lexicon will be ignored by our classification models, as they are entirely dependent on the dataset being used and the dataset of the British Isles we rely on does not expose dialect-specific lexicons.

3.1.2 Architectures

The **baseline** model for speaker and dialect accent embeddings this thesis adopts is based on the **d-vector** concept. A d-vector is simply a way to refer to speaker embeddings generated by a DNN, hence the *d* prefix. The standard way to compute such d-vectors, as described in [181] and shown in Figure 3.2, is through a stack of LSTM layers processing spectrogram segments. In particular, the full spectrogram of shape $[B, M, T]$ (with B batch size, M number of mel-bands and T number of time steps) is unfolded in a sequence of overlapping tensors of shape $[B, M, S]$, where S is the segment length. Then,

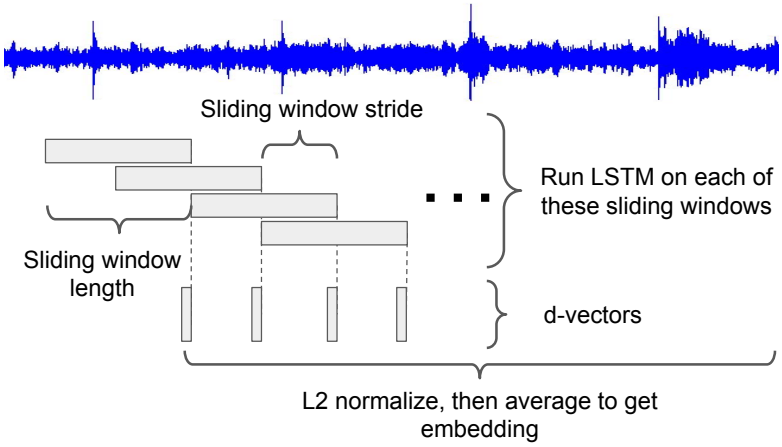


Figure 3.2: Computation of d-vector embeddings [181] in the baseline architecture. Intermediate representations are computed in a sliding window fashion and averaged to obtain a final embedding.

each segment is fed into a recurrent module and hidden states are collapsed in a single dimension by either averaging or simply by keeping the last one. Collapsed vectors are then projected onto the embedding size and segment embedding vectors are averaged together to obtain the embedding vector of the full spectrogram.

The standard d-vector architecture is quite simple, but effective. Still, the major strength of the d-vector model comes from its **loss function**, namely Generalized End-To-End (GE2E) loss, first introduced in [181]. Considering a mini-batch of size $M \times N$ with M utterances from each of N different speakers and $e_{j,i}$ as the embedding of utterance i ($1 \leq i \leq M$) from speaker j ($1 \leq j \leq M$), the GE2E loss is computed as reported in Equation 3.1.

$$L_{GE2E} = -\frac{1}{N} \sum_{j,i} \log \frac{\exp(S_{j,i,j})}{\sum_{k=1}^{N} \exp(S_{j,i,k})}, \tag{3.1}$$

where

$$S_{j,i,k} = \begin{cases} w \cdot \cos(e_{j,i}, c_j^{(-i)}) + b & \text{if } k = j \\ w \cdot \cos(e_{j,i}, c_k) & \text{otherwise} \end{cases}, \tag{3.2}$$

with w and b learnable weights and biases and centroids c computed as in Equation 3.3.

$$\begin{aligned} c_j &= \tfrac{1}{M} \textstyle\sum_{m=1}^{M} e_{j,m}, \\ c_j^{(-i)} &= \tfrac{1}{M-1} \textstyle\sum_{\substack{m=1 \\ m \neq i}}^{M} e_{j,m}. \end{aligned} \tag{3.3}$$

At each training step, GE2E builds a similarity matrix S (as shown in Figure 3.3) that defines the similarities between each $e_{j,i}$ and all centroids c_k. This approach contrasts

with TE2E [65], a previous work by the same authors, where similarity is a scalar value that defines how close embedding vector $e_{j,\sim}$ is to a single tuple centroid c_k.

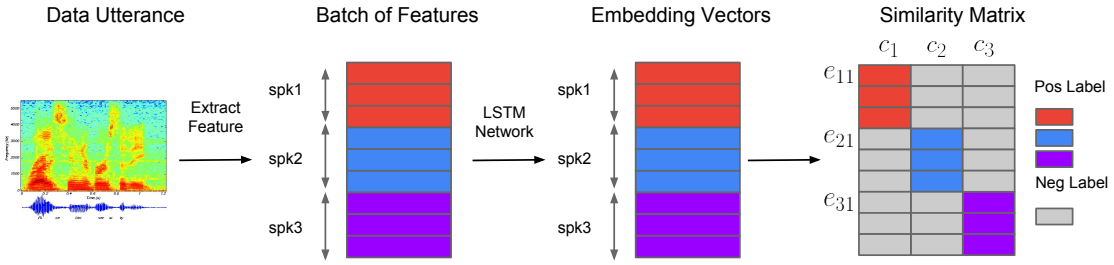


Figure 3.3: Similarity matrix with the GE2E loss [181]. For all speakers, the same number of utterances is added to a mini-batch and the similarity matrix between d-vectors is used as a mean for contrastive learning.

Other than the baseline d-vector model, we rely on **TitaNet** [93], an encoder-decoder architecture, shown in Figure 3.4, that at the time of writing is SOTA on the speaker verification and diarization tasks.

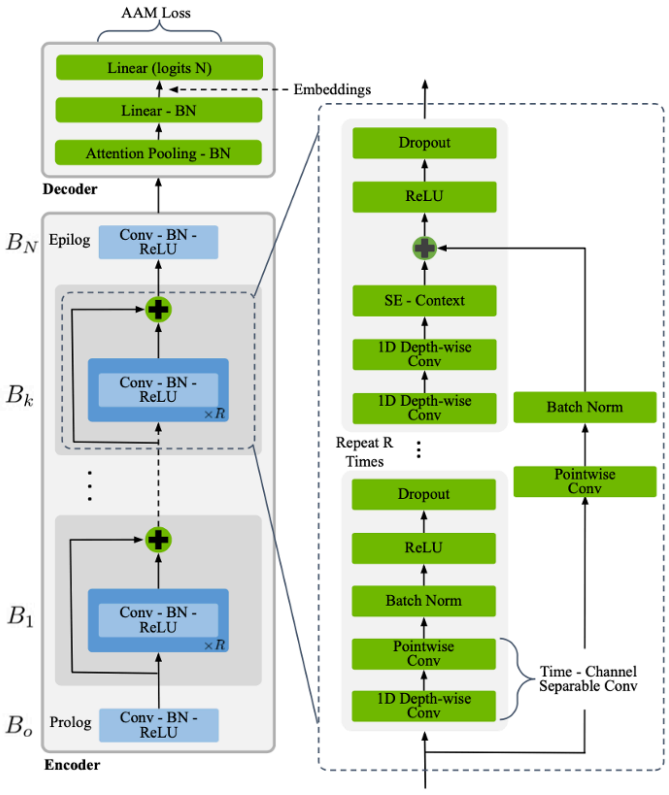


Figure 3.4: TitaNet architecture, composed of an encoder-decoder structure and an attentive statistics pooling layer to compute utterance-level features.

Most of TitaNet's complexity can be attributed to the encoder, while the decoder's number of parameters is kept low. The former is divided into blocks $B_0, \ldots, B_N$, where

B_0 is called *prolog* and B_N is called *epilog*. The prolog and epilog blocks are a sequence of 1D convolution, Batch Normalization (BN) [74] and ReLU activation, while mega-blocks are composed by R sub-blocks each ($R = 3$ in the TitaNet paper).

One sub-block is a sequence of 1D depth-wise convolution, BN, ReLU and dropout [159]. After an input passes through all sub-blocks, it also goes through a Squeeze-and-Excitation (SE) [70] layer. Then, the initial mega-block input is merged with the SE output through a skip connection that contains a 1×1 convolution (to match input and output channels) and BN. After the skip connection, ReLU activation and dropout are applied and one mega-block computation is over.

The decoder has an **attentive statistics pooling** layer [130] to form fixed-size utterance-level features from variable frame-level features. It can be thought of as an alternative to global average pooling [110], where each channel is given different weight, computed by the attention mechanism. The main peculiarity of attentive statistics pooling is that it generates not only weighted means but also weighted variances and concatenates them to form a single attention context vector.

Pooling is a mandatory step in audio embedding models, as the goal is to compress frame-level features to **utterance-level features**. In order to understand why that is the case, looking at input/output shapes helps. In the case of TitaNet, its input is a mel-scaled spectrogram of shape $[B, M, T]$ (with B batch size, M number of mel-bands and T number of time steps) and its output is an embedding matrix of size $[B, E]$, where E is the chosen embedding size. Figure 3.5 shows the shape adjustments that an input tensor has to go through in order to get the correct output: the encoder produces an output of shape $[B, H, T]$ (where H is the hidden size, a hyper-parameter), that has one too many dimensions in order to be used as an embedding vector.

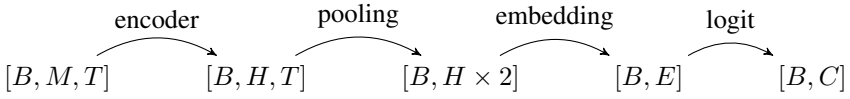


Figure 3.5: Tensor shape journey in TitaNet. The input mel-spectrogram of shape $[B, M, T]$ flows through the network and gets mapped to an embedding vector of shape $[B, E]$.

So, one solution is to compress the time-dimension (i.e. the last one). The simplest approach for compression would be to average or sum along the dimension that has to be removed [158], but in this way, the same weight would be given to all spectrogram frames. Instead, it is often the case that some frames are far more important than others to discriminate speakers and/or dialect accents. To account for such variability, the attention

mechanism comes to the rescue, as it allows to compute frame-level importance as part of the training procedure. As described in [130], a simple **attention model** computes a scalar score e_t for each frame-level feature, i.e. $e_t = v^T f(Wh_t + b) + k$, where $f(\cdot)$ is a non-linear activation function, such as $\tanh$ or ReLU. The score is normalized over all frames by a softmax function to get $\alpha_t = \frac{\exp(e_t)}{\sum_k^T \exp(e_k)}$. The normalized score α_t is then used as the weight in the pooling layer to calculate the weighted mean vector $\mu = \sum_t^T \alpha_t h_t$.

Okabe, Koshinaka, and Shinoda additionally compute **higher-order statistics** in the attention module, where standard deviations are calculated as $\sigma = \sqrt{\sum_t^T \alpha_t h_t \odot h_t - \mu \odot \mu}$ and $\odot$ is the Hadamard product. The intuition behind including weighted standard deviations is to bring even more discriminability to utterance-level features and model long-term variations in accord with importance. Finally, weighted means and standard deviations are concatenated to get an output of shape $[B, H \times 2]$. Notice that the additional number of parameters when using attentive statistics pooling does not come from the attention module itself, as both μ and σ share the same weights α, but from subsequent layers that have to work with an input double the size of what would be obtained with the non-statistics version of attentive pooling.

After the attention mechanism, a linear layer projects vectors from the attention context output size to the given embedding size (192 in the TitaNet paper). This linear layer is followed by BN and its output is the embedding vector.

Afterwards, another linear layer maps vectors from the embeddings space to output logits and the chosen loss function is applied. Candidate loss functions include standard CE or angular margin variants of it. In its original implementation, TitaNet relies on an **additive angular margin loss**, also referred to as ArcFace loss [43] and computed as in Equation 3.4.

$$L_{ArcFace} = -\frac{1}{N} \sum_{i=1}^{N} \frac{\exp(s \cdot \cos(\theta_{y_i,i} + m))}{\exp(s \cdot \cos(\theta_{y_i,i} + m) + \sum_{j \neq y_i} \exp(s \cdot \cos(\theta_{j,i}))} \tag{3.4}$$

In Equation 3.4, s is the scale parameter, m is the additive margin, N is the total number of utterances in the mini-batch, y_i ($1 \leq y_i \leq C$) is the label for utterance i ($1 \leq i \leq N$) and C is the total number of classes. Figure 3.6 shows the flow of operations needed to compute the ArcFace loss on an example input tensor x_i.

TitaNet's architecture is heavily inspired by previous work. In particular, the main building blocks were first introduced in QuartzNet [97], an ASR model that was SOTA

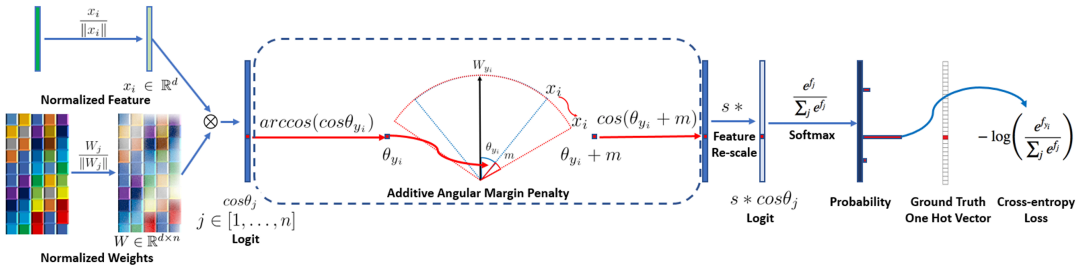


Figure 3.6: ArcFace loss computation. The margin m is summed to the angle between normalized features and weights.

in 2019; then, SE was brought in by ContextNet [63], another ASR architecture, taking the lead in SOTA over QuartzNet in 2020. The same authors presented another speaker recognition/verification model 1 year before TitaNet, i.e. SpeakerNet [92]. The main difference between SpeakerNet and TitaNet is that the former is based on QuartzNet, while the latter is based on ContextNet. Moreover, SpeakerNet uses MFCCs input features, while TitaNet relies on mel-scaled spectrograms. Another gap between the 2 models is in the amount of data used: SpeakerNet was trained with VoxCeleb1 [127] and VoxCeleb2 [28] public datasets, while TitaNet on the same, plus Fisher [27], Switchboard [58] and LibriSpeech [134] datasets.

3.1.3 Evaluation

In terms of evaluation, speaker verification systems are assessed through Receiver Operating Characteristic (ROC) curves and derived metrics. Furui explains how metrics such as Equal Error Rate (EER) and minimum of the Detection Cost Function (DCF) are computed. As with all binary classification problems, in speaker verification (and dialect verification) there are 4 different error types, as depicted in the confusion matrix in Figure 3.7. In particular, denoting P as the positive class (which marks the speaker/dialect of the utterances being compared as the same) and N as the negative class, the outcome of a prediction either coincides with the target, in which case a true positive TP or a true negative TN is spawn, or it does not, in which case the count of false positives FP or false negatives FN increases.

Switching from raw numbers to rates, we obtain $TPR = \frac{TP}{P}$, $FPR = \frac{FP}{P}$, $TNR = \frac{TN}{N}$, $FNR = \frac{FN}{N}$. In the speaker verification literature, FNR is also referred to as FR, i.e. the probability of false rejection, while FPR is also referred to as FA, i.e. the probability of false acceptance.

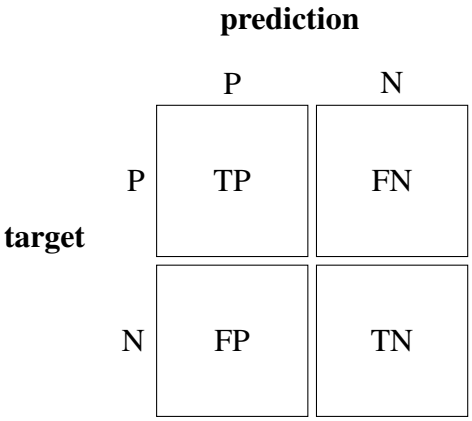


Figure 3.7: Confusion matrix in a binary classification problem. P represents the positive class, N the negative one and T and F indicate True and False prefixes.

Plotting FPR over TPR gives rise to the ROC curve. Figure 3.8 shows an ROC plot comparing three speaker verification systems A, B and C where only the decision threshold a, b and c is varied: such curves show system B as being consistently superior to the others.

The ROC curve is sufficient to compute the EER score of each system: in Figure 3.8, EER is the point at the intersection of each curve with the dashed straight line of 45 degrees.

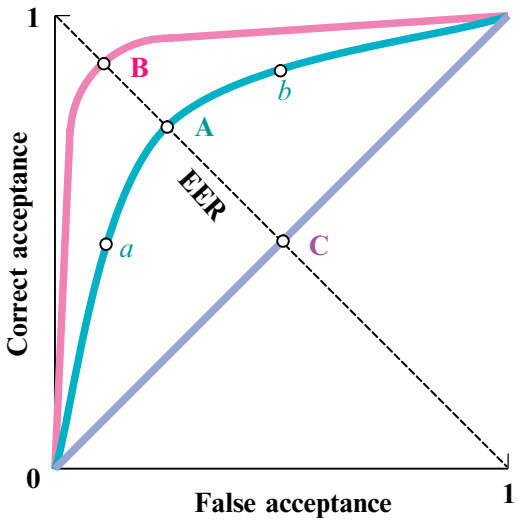


Figure 3.8: ROC curves comparing speaker verification systems. A, B and C refer to the same system with different decision thresholds a, b and c.

Another popular metric in speaker verification evaluation is the minimum of the DCF. Equation 3.5 reports its computation, following the standard introduced in the NIST

speaker recognition evaluations [140].

$$\min DCF = \frac{\min C_{DET}}{\min\{C_{FR} \cdot P_C, C_{FA} \cdot (1 - P_C)\}} \tag{3.5}$$

Computing $\min DCF$ requires obtaining $\min C_{DET}$ first, and then normalizing it. To compute $\min C_{DET}$, the minimum C_{DET} at different decision thresholds $\theta = [\theta_1, \ldots, \theta_N]$ is selected. Equation 3.6 reports a C_{DET} computation for a generic threshold θ_i.

$$C_{DET}(\theta_i) = C_{FR} \cdot FR(\theta_i) \cdot P_C + C_{FA} \cdot FA(\theta_i) \cdot (1 - P_C) \tag{3.6}$$

Computing C_{DET} and $\min DCF$ also requires setting the following hyper-parameters: the cost of a false rejection C_{FR}, the cost of a false acceptance C_{FA} and the a priori probability of a customer P_C. In [140], such variables are set to $C_{FR} = C_{FA} = 1.0$ and $P_C = 0.01$ and this thesis follows the exact same approach. Finally, Figure 3.9 shows examples of Detection Error Trade-off (DET) curves[3].

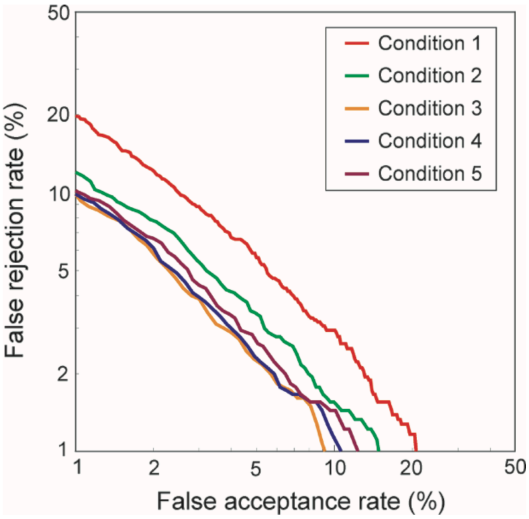


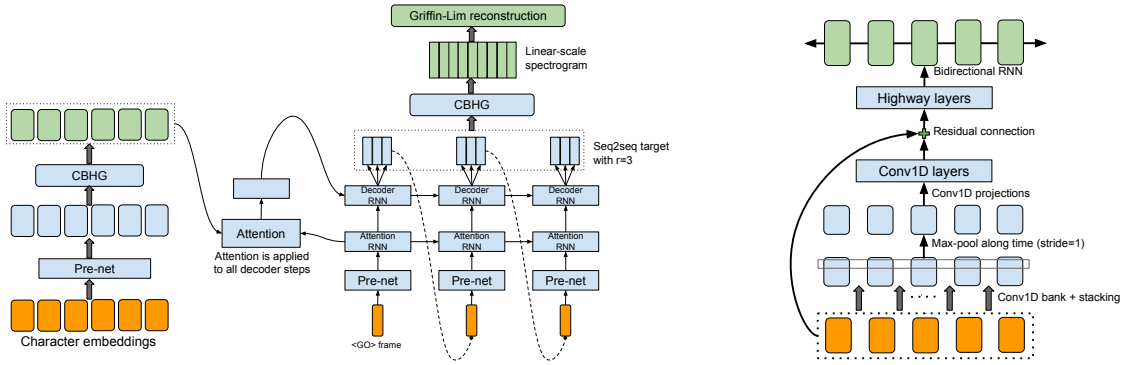
Figure 3.9: Examples of DET curves for different systems. The EER for each system is obtained by intersecting its DET curve with the first bisector.

[3] EER corresponds to the intersection of the DET curve with the first bisector curve [56].

3.2 Acoustic model

This section deals with the introduction of the core module that is tasked to convert input text into speech representations. As acoustic models are part of the standard TTS pipeline described in Section 2.3, no more words will be spent in describing the goals and objectives of such models. Rather, the following sections report the neural architecture selected for this thesis and delineate well-established evaluation methods and metrics for acoustic models, both as a whole and for what concerns this thesis.

3.2.1 Architectures



(a) Encoder-decoder architecture.

(b) CBHG module.

Figure 3.10: The Tacotron model [185], in its first iteration. The architecture (left) is a S2S one and comprises a set of CBHG modules (right).

The acoustic model of our choice is that of Tacotron 2. Starting from its first version, **Tacotron** [185] is a S2S model [8, 161] that directly generates spectrograms from raw characters. This approach greatly simplifies the legacy TTS pipeline described in Subsection 2.3.1 and achieves better performance than SPSS-based acoustic modelling, while also enabling rich and easy conditioning. Tacotron slightly modifies the standard S2S approach, but still maintains its basic components:

1. **Encoder**: the combination of two custom modules, namely pre-net and CBHG, the former being a bottleneck block (composed of linear, ReLU and dropout layers) and the latter being a contrived building block (shown in Figure 3.10b) made of convolutions, batch normalization, highway and GRU cells (hence the name); the output of this step is a hidden representation of the whole input sequence;

2. **Decoder**: a number of GRU cells are given in input a vector obtained by concatenating outputs from a "pre-net" (the same module used in the encoder) and the dynamic context vector produced by the attention mechanism described in [8]; the output of this step is an 80-band mel-scaled spectrogram;

3. **Post-processing**: the combination of a CBHG module and a final linear layer to convert mel-scaled spectrograms to linear ones and perform waveform synthesis through the Griffin-Lim algorithm.

About the **encoder**, its output should be a set of semantically-meaningful hidden states, one for each character/phoneme in the input sequence. Intuitively, convolutional layers are there to encode local correlations between inputs and to enable the inclusion of a relatively large context, in ways comparable to n-grams processing.

Once the encoder outputs are generated, hidden states are fed to an **attention network** that consumes them to generate context vectors. In a plain S2S model, the encoder would map the entire input sequence (such as a sentence) in a single vector (named **context vector**), that would ideally correctly summarize the input. Then, the decoder would be trained to predict the output sequence, one step at a time, using the same context vector at each step. Bahdanau, Cho, and Bengio extend the concept of context vector in the following way: each decoder step relies on a different context vector, by learning to attend to a mix of encoder hidden states. This attention mechanism removes the tight information bottleneck that was imposed by the use of a single, fixed-length context vector. As a side effect, this attention mechanism allows the model to align outputs with corresponding inputs in an effective way: in acoustic modelling, this means that the models have an intrinsic way of learning grapheme/phoneme durations. Formally, in **Bahdanau attention** (or content-based attention) the context vector for decoder step i is computed as $c_i = \sum_j \alpha_{i,j} h_j$, where h_j is the encoder hidden state at step j and $\alpha_{i,j}$ is a learnable attention weight (also referred to as alignment), that encodes the relevance of encoder step j for the decoder step i. The soft attention weights $\alpha_{i,j}$ are computed as $\alpha_{i,j} = \frac{\exp(e_{i,j})}{\sum_k \exp(e_{i,k})}$, where $e_{i,j}$ is a score (referred to as energy) obtained as $e_{i,j} = v^T f(W s_{i-1} + U h_j)$, with v, W, U learnable weights, s_{i-1} the decoder output from the previous step and f being a non-linear function such as $\tanh$.

As soon as a context vector is generated for the current step, the **decoding stage** can start. At each decoder step, the previous spectrogram frame (or a special <GO> at the very beginning) is passed through a pre-net module and its output is concatenated with

the attention context vector, to then be fed to a simple RNN. The recurrent network is finally tasked to output the predicted spectrogram frame(s): how many to predict at each step depends on the selected **reduction factor** parameter r[4]. Something to keep in mind is that being an auto-regressive S2S model, Tacotron suffers from a train-test mismatch, as **teacher forcing**[5] [190] is used at training time, while auto-regressive generation has to be enabled at inference time.

As far as losses go, Tacotron only relies on reconstruction errors. In particular, its **total loss** is given by an equally-weighted sum of a simple L_1 loss for both the decoder outputs (mel-scale spectrograms) and post-processing net (linear-scale spectrograms).

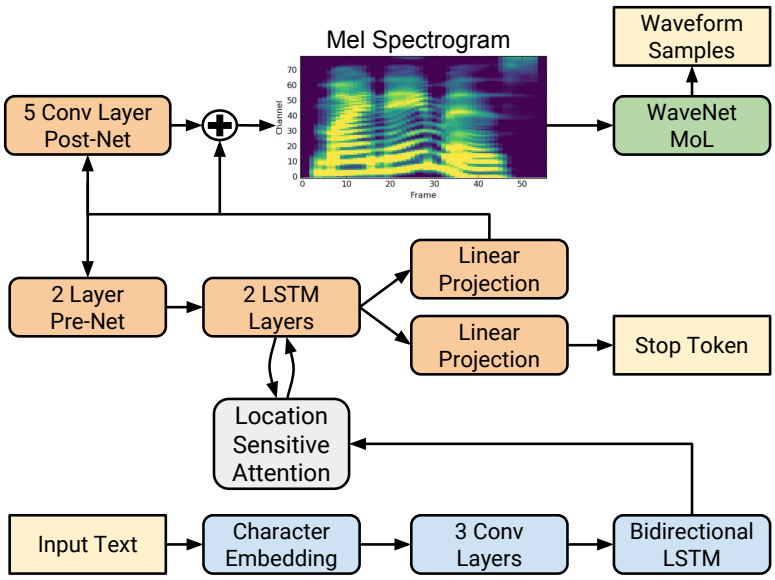


Figure 3.11: The Tacotron 2 architecture [152]. The CBHG module of the first variant is replaced for simpler convolutional blocks and the attention mechanism is location-sensitive.

The second version of Tacotron was presented 1 year after the first one, by the same authors. **Tacotron 2** enhances the original model through small but effective changes, mainly related to architectural revisions, as shown in Figure 3.11. For what concerns the encoder, it is greatly simplified as it only contains 1D convolutions (followed by batch normalization and ReLU activations) and GRU cells are replaced by LSTM ones. Experiments on switching the Tacotron 2 encoder with its Tacotron counterpart (and

[4] Wang et al. report that predicting multiple, non-overlapping output frames at each decoder step is beneficial for the model to learn alignments faster, likely because neighbouring speech frames are correlated and each character usually corresponds to multiple frames. This also allows for more efficient training and inference procedures, given that predicting r frames at once divides the number of decoder steps by r.

[5] Teacher forcing [190] is a technique to train auto-regressive models, whereby at training time the model is fed with the ground truth output sequence for the previous step, instead of being given what it produced one step before.

vice-versa) result in very similar outcomes, thus suggesting that the former has enough capacity to correctly perform its task.

Moreover, a new **location-sensitive** attention module [26] is introduced in place of the one described by Bahdanau, Cho, and Bengio. In location-sensitive attention the energies $e_{i,j}$ are computed as $e_{i,j} = v^T f(V l_{i,j} + U h_j)$, where $l_{i,j}$ are the location features obtained by convolving previous alignments α_{i-1} with convolutional filters F and v, V, U are learnable weights. At the opposite end of the spectrum w.r.t. content-based attention, location-based attention doesn't care at all about the content of the input tokens, but only cares about their locations and the distances that exist between them. To get the best of both worlds, an hybrid between the content and location based modules can be used, where energies are obtained as $e_{i,j} = v^T f(W s_{i-1} + V l_{i,j} + U h_j)$. Shen et al. rely on a slightly modified version of this **hybrid attention**, that uses cumulative attention weights from previous decoder time steps as an additional feature.

Also, the final *post-net* ditches CBHG for a 5-layer CNN with BN and residual connections. As a final (and most important) remark, waveform synthesis is done using a modified version of **WaveNet** [131] (as opposed to the Griffin-Lim algorithm) that takes mel-spectrograms as input, instead of the original set of features (linguistic features, predicted log fundamental frequency and phoneme durations). The feature prediction network and the modified WaveNet are trained separately, one after the other, with the first one being trained with a Mean Squared Error (MSE) loss (as opposed to L_1 loss used in Tacotron) between the predicted and target mel-spectrograms. Experiments show that Tacotron 2 is able to generate natural speech that is sometimes indistinguishable from human speech.

For what regards the **multi-speaker** extension of the acoustic model, we follow the architecture depicted in Figure 3.12, where the speaker encoder is used to condition the synthesis network on a reference speech signal from the desired target speaker [77]. The conditioning happens by concatenating an embedding vector for the target speaker with the encoder's output at each time step. Alternative solutions include substituting the speaker encoder for a lookup table [7, 139], with which the model learns a fixed embedding for each speaker in the training set, thus completely disposing of zero-shot abilities. Also, extensive research has been carried out on the best position to concatenate speaker embeddings. For example, Cooper et al. show that concatenating speaker embeddings

to both the pre-net and encoder outputs could be beneficial in terms of speaker similarity while concatenating on the same plus at the *post-net* stage has been observed to be detrimental.

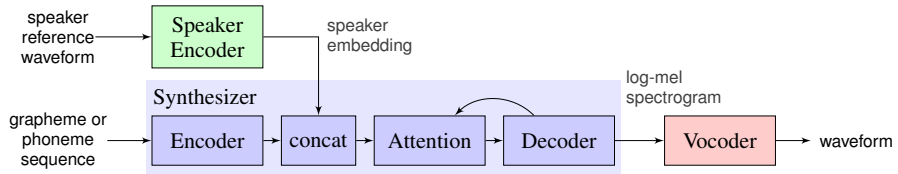


Figure 3.12: From speaker verification to multi-speaker Tacotron 2 [77]. Speaker embeddings pre-trained on the speaker verification task are used to condition the acoustic model on speaker identity.

Switching to the more challenging multi-speaker and **multi-dialect** scenario, the reasoning follows the work done for multi-speaker and multi-lingual models. In particular, the first work to introduce cross-lingual synthesis with a Tacotron 2 architecture relied on a combination of speaker and language embeddings, as shown in Figure 3.13, with additional optimizations that are considered out of scope for this thesis. What's important is that Zhang et al. concatenate language embeddings in the same position as speaker embeddings. This can be done when the acoustic model input is in the form of language-dependent phonemes or graphemes, i.e. different input mappings are used for each language. Instead, the usage of shared representations, such as a single set of raw characters (as in the case of dialects), forces language (or dialect) embeddings to be concatenated to each character embedding, before being fed to the encoder.

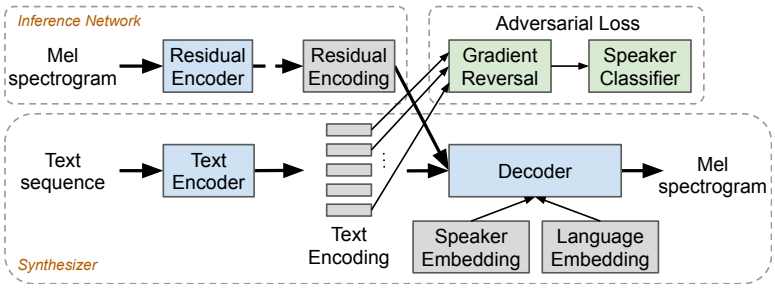


Figure 3.13: Cross-lingual Tacotron 2 [200]. Pre-trained speaker and language embeddings are used to enable multi-speaker cross-lingual synthesis.

In this thesis, we rely on a custom implementation of Tacotron 2, with the possibility to condition both on speaker and dialect accent embeddings. No experiments were carried out with the first version of Tacotron, but its description is still reported, as we consider it to be an integral part of its successor.

3.2.2 Evaluation

TTS systems are usually evaluated along 2 dimensions: **naturalness** and **intelligibility**. Naturalness refers to how close the synthesized speech is to real-life speech, and intelligibility refers to the ease of understanding the spoken content. Though, other dimensions might be considered depending on the application. For example, in this thesis, we mainly focus on **accentedness**, where the goal is to evaluate speech in terms of how close it sounds to a native speaker of a specific dialect.

There are 2 main ways to evaluate TTS systems, i.e. either using subjective (or perceptual) tests or through objective metrics. In the literature, the former evaluation type is far more popular than the latter and this is because most of the objective metrics used so far do not correlate well with human perception, as in all generative models [171]. For example, one of the standard approaches relied upon in a DL scenario is to split the whole dataset into training and validation sets, with the latter functioning as a model selection mechanism, i.e. the "best" model can be chosen as the one having the least training-objective loss on this unseen validation split. Unfortunately, in the case of auto-regressive TTS, it's not always the case that better validation metrics imply having a better model [116]. Hence, researchers tend to rely on visual comparisons (e.g. of the produced mel-spectrograms) and **informal listening** sessions to select the best checkpoints.

Formal **subjective evaluations**, other than informal listening, include crowd-sourced tests, such as Mean Opinion Score (MOS) and MUltiple Stimuli with Hidden Reference and Anchor (MUSHRA). Both tests are audio-only listening tests, where participants evaluate unrelated, context-free utterances.

MOS is usually used to obtain an absolute number identifying the performance of a single model. In MOS each listener is given a set of samples that have to be classified on a scale of 1 to 5, with 1 corresponding to "bad" and 5 meaning "excellent". The evaluation has to be preceded by a meaningful question that depends on what property of speech has to be evaluated, e.g. *"Please listen to the sample and judge using a five-point scale their ... quality/naturalness/similarity to a certain speaker"*. The MOS score for each sample is then computed as the average over single ratings, while the MOS score for the model that produced the samples is the arithmetic mean over each sample's MOS. Assuming each of the N samples to be rated by M different listeners, the MOS score for the model

is computed as in Equation 3.7, with $r_{j,i}$ being the rating of sample i from listener j.

$$MOS = \frac{1}{N}\sum_{i=1}^{N}\frac{\sum_{j=1}^{M} r_{j,i}}{M} \tag{3.7}$$

Instead, **MUSHRAs** are used to perform a relative comparison between multiple models. In MUSHRA each listener is given a set of parallel examples, i.e. the same text prompt synthesised with all the models being evaluated, along with a labelled reference, a hidden version of the reference and one or more anchors, and they all have to be positioned on a scale of 0 to 100 (as shown in Figure 3.14). The reference is usually the ground-truth speech (i.e. the recorded prompt) and acts as an upper bound, while the purpose of the anchor(s) is to make the scale be closer to an "absolute scale", making sure that minor artefacts are not rated as having very bad quality [189]. In TTS evaluations, usually, only one anchor is used and it corresponds to speech synthesised by a baseline model, i.e. a system which should consistently be ranked as the worst. The benefits of MUSHRA over MOS are that it requires fewer participants to obtain statistically significant results[6] and the 0-100 scale makes it possible to rate very small differences. The main downside of MUSHRAs is that it takes a long time for listeners to complete the entire test, thus implying greater costs to be incurred.

In the case of 2 models being evaluated, an alternative to MUSHRA is the use of **preference tests** (also called A/B tests), where listeners are presented with two parallel speech samples (one from model A and one from model B) and asked to indicate which one has more of a certain property. In a preference test, the question would be *"Which sample do you prefer?"*, but it could also be *"Which sample sounds more natural?"* or *"Which sample sounds closer to a native Irish speaker?"*. The choices can be forced, i.e. either "A" or "B", or they can include a neutral option (e.g. "no preference") when both systems sound more or less the same. Preference tests are, on average, much more affordable than MUSHRAs, but the interpretation of their results tends to be more difficult.

For what regards **objective metrics**, intelligibility can be measured with the aid of an ASR model [21]. The standard procedure is the following: the text is given as input to the TTS model being evaluated and its output is given as input to a pre-trained ASR model that produces text from a speech representation; then, input and output texts are compared to assess intelligibility. Different metrics span depending on how fine-grained the

[6] MUSHRAs require fewer participants than MOS because all systems are presented at the same time, on the same samples (sentences), so that a paired t-test can be used for statistical analysis [189].

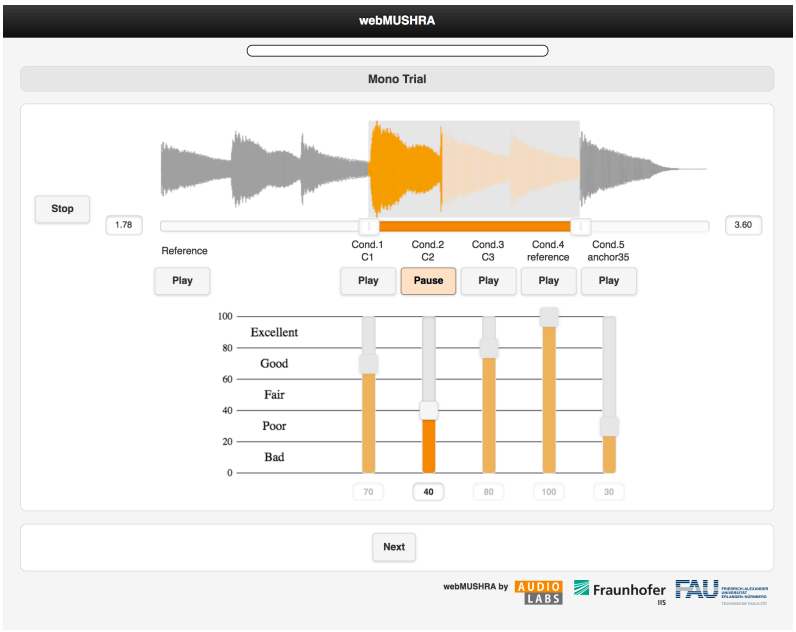


Figure 3.14: Example of a MUSHRA test with the webMUSHRA interface [151]. $C1$, $C2$ and $C3$ are the compared systems, while *reference* is ground-truth and *anchor35* is a baseline model.

comparison methodology is: looking at the character level gives rise to the Character Error Rate (CER) measure, comparing phonemes gives rise to Phoneme Error Rate (PER), while word-level mismatches are summarized by the Word Error Rate (WER) metric. The disadvantage of using an ASR model for intelligibility measures is the propagation of errors, i.e. the quality of the chosen ASR model has a great impact on the downstream evaluation of the TTS system.

Other approaches have been explored to predict properties such as intelligibility and naturalness. For example, neural models have been trained to directly estimate MOS scores for the synthesized audio. The most popular models in this field are **MOSNet** [112] and its successors, such as MBNet [105] and LDNet [71]. Such models are trained on datasets of synthesized speech and their corresponding MOS score; however, the collection of such datasets for different languages and objectives is not trivial and results are not yet on par with (or even close to) crowd-sourced evaluations.

Promising neural approaches include Fréchet Audio Distance (FAD) [81], which is an adaptation of Fréchet Inception Distance (FID) [68] from the generative image domain to the generative audio domain, initially proposed for music enhancement algorithms and later used in TTS applications [12]. Kilgour et al. show that FAD correlates more closely to human perception than signal based metrics, such as Signal to Distortion Ratio

(SDR), cosine distance and magnitude L_2 distance. Other signal-based metrics include Perceptual Evaluation of Speech Quality (PESQ) [146], Mel-Cepstral Distortion (MCD) [98] and many more.

Thus, despite the fact that TTS systems are ubiquitous nowadays, their evaluation it's still a controversial topic, as it gets approached in more or less the same way as in the late 1990s [180]. Many researchers are now studying ways to bridge the gap between human perception and objective metrics, using techniques such as the ones explained above, but much work still needs to be done.

3.3 Voice/accent conversion system

Conversion systems in the audio domain have the aim of transforming one or more speech properties while keeping all the others intact. In this thesis, audio conversion systems are used in 3 different ways: (i) to convert speaker identity, (ii) to convert dialect accent information, while keeping speaker identity the same and (iii) to convert both speaker identity and dialect accent information. The following sections present the main tasks and terminology used in audio conversion systems, along with the introduction of the main model used for conversion purposes in the pipeline shown in Figure 3.1. In terms of evaluation metrics and techniques, we refer the reader to what has already been detailed, as acoustic models and conversion systems share the same output space and are thus evaluated in comparable ways.

3.3.1 Tasks and terminology

One of the most popular tasks in audio conversion systems is that of **Voice Conversion**. The ultimate goal of a VC system is to convert an input utterance by a source speaker into an output utterance by a target speaker. This means that the output of a good VC system should be the target speaker speaking the linguistic content found in the input utterance. Thus, a VC system should learn to disentangle speaker identity, along with its prosody information (such as speaking rate), from linguistic content. VC systems are usually subdivided in many categories, one of which being **parallel and non-parallel**. The former indicates a VC system where all speakers in the dataset speak the same prompts, while the latter is used for the more generic variant of a VC system trained with data such that each speaker can speak different input text. Another important class of VC systems is related

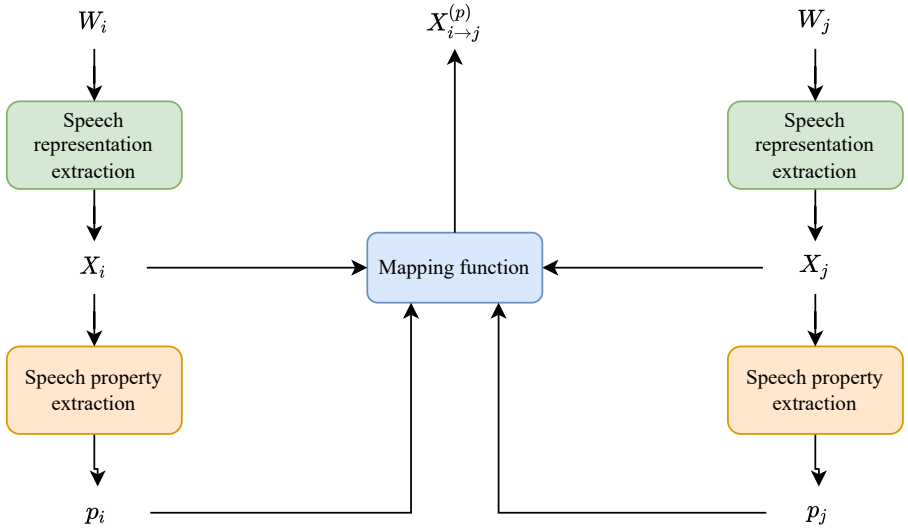


Figure 3.15: Audio conversion system, seen as a function that maps between speech properties in the same domain.

to the amount of targeted speakers and the overall scope of the model: a VC system is usually indicated as ***x*-to-*y***, where x and y can take values in {one, many, any}. The x indicates which source speakers the VC system can convert from and the y indicates which target speakers the model can convert to. If x (y) equals "one", it means that the VC system only supports a single source (target) speaker, if x (y) equals "many", then the VC system supports all source (target) speakers in the training set, while if x (y) equals "any", all source (target) speakers are supported (also those that were unseen during training). With this in mind, the simplest VC system would be a parallel one-to-one model, while the most ambitious one would be its non-parallel any-to-any counterpart. It has to be noted that not all literature follows the same definition in terms of x-to-y, with some work using the terms "many" and "any" interchangeably. The last, but not least, important class of VC systems is **few-shot and zero-shot**: a VC model has few-shot capabilities if it can successfully perform conversion when only a handful of utterances from the source or target speaker were seen during training, while a zero-shot model can do the same even with zero utterances. So, few-shot models are subsets of zero-shot models, meaning that a model that can perform zero-shot conversion can usually work well in a few-shot fashion.

Another application of audio conversion systems is in the task of **Accent Conversion**, which seeks to transform only those features of an utterance that contribute to accent while maintaining those that carry the identity of the speaker. In the literature, the

term AC is usually used to indicate a more specific task, namely Foreign Accent Conversion (FAC), where the goal is to create a new voice that has the voice identity of a given second-language speaker but with a native accent [46]. There are multiple practical applications of FAC, such as Computer-Assisted Pronunciation Training (CAPT) [53], in which a language learner is provided with ground truth speech of themselves speaking with native accent in the target language: this can be very useful for the language learner to correct pronunciation mistakes and articulation issues that are due to speaking habits and biases towards their native language. In this thesis, we refer to AC as the more fine-grained task of converting regional accents.

Figure 3.15 shows a general representation of audio conversion systems. Here, X_* is a speech representation such as a mel-spectrogram and p_* is a vector representation of the property of speech the system aims to convert (in VC this would be speaker identity, while in AC it would be accent). Subscripts i, j in X_* and p_* indicate source and target utterances, respectively. As shown in Figure 3.15, the audio conversion system C takes as input X_i and X_j, extracts information p_i and p_j and outputs a speech representation $X^{(p)}_{i\to j}$, i.e. $C^{(p)}(X_i, X_j) = X^{(p)}_{i\to j}$.

3.3.2 Architectures

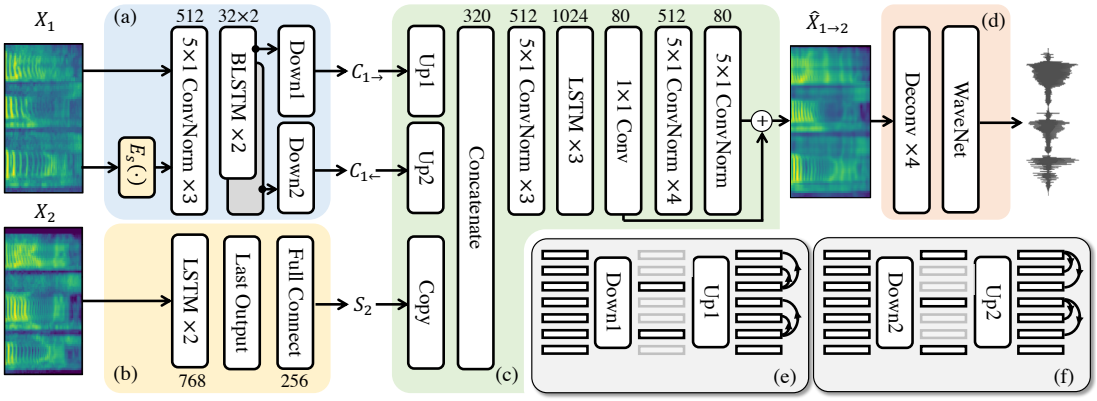


Figure 3.16: AutoVC architecture [142]. X_1 is the source utterance and X_2 is a snippet of speech from the target speaker. The goal is to obtain a representation $\hat{X}_{1\to 2}$ with the linguistic content of X_1 and the speaker identity of X_2.

AutoVC is the main architecture we rely on for both voice and accent conversion. **AutoVC** [142] is a an autoencoder-based VC system, working in the non-parallel many-to-many setting, as well as the zero-shot one. The main peculiarity of AutoVC is that

it achieves style transfer with a simple autoencoder architecture, without making use of either sophisticated auxiliary and adversarial losses, as done in many Generative Adversarial Network (GAN)-based [60] VC systems, or the maximization of a variational lower bound of the output probability, as required by Conditional Variational Auto Encoder (CVAE) models [86, 87].

Being an **autoencoder**, AutoVC is composed of two main modules, an encoder and a decoder. The former takes care of mapping the input to a low-dimensional space, also referred to as the latent space, while the latter attempts to reconstruct the original input starting from its latent representation. This learning procedure is an instance of the more general information bottleneck method [172], in which the goal is to squeeze the information that an input signal provides about another signal through a limited set of codewords.

AutoVC makes use of the **information bottleneck** method by nudging the latent vectors to encode only linguistic content and not source speaker timbre. Then, in order to perform the conversion, target speaker information is concatenated to the latent code (which is assumed to only represent linguistic features) and decoded into a speech representation. AutoVC relies on a simple, yet effective, trick to discourage the latent space to encode speaker information, i.e. speaker embeddings are concatenated at each time-step of the input speech representation. In this way, assuming a carefully tuned bottleneck, the encoder is forced to lose some information and the easiest way to achieve perfect reconstruction would be to simply discard speaker information, as already supplied in the input.

This strategy is powerful but very dependent on the chosen **hidden size**. When the bottleneck is too wide, source speaker information leaks through the bottleneck and converted speech may sound exactly the same as source speech. On the other hand, when the bottleneck is too narrow, speaker information does not make its way to the latent space, but also linguistic features might do the same and get lost, i.e. converted speech would sound as if it was uttered by the target speaker, but it would present mumbling issues, word skipping and other content-related errors.

AutoVC works in two different ways at train and test time. At **training time**, the source speaker is the same as the target speaker and the model learns to reconstruct the input signal. At **inference time**, source and target speakers are different and the output should contain no information about the source speaker. In terms of training objectives,

AutoVC only relies on a combination of self-reconstruction and content code reconstruction losses, as reported in Equation 3.8.

$$\begin{aligned} L &= L_r + \lambda L_c \\ L_r &= \mathbb{E}[\|X_{i\to i}^{(p)} - X_i\|_2^2] \\ L_c &= \mathbb{E}[\|E(X_{i\to i}^{(p)}) - E(X_i)\|_1] \end{aligned} \tag{3.8}$$

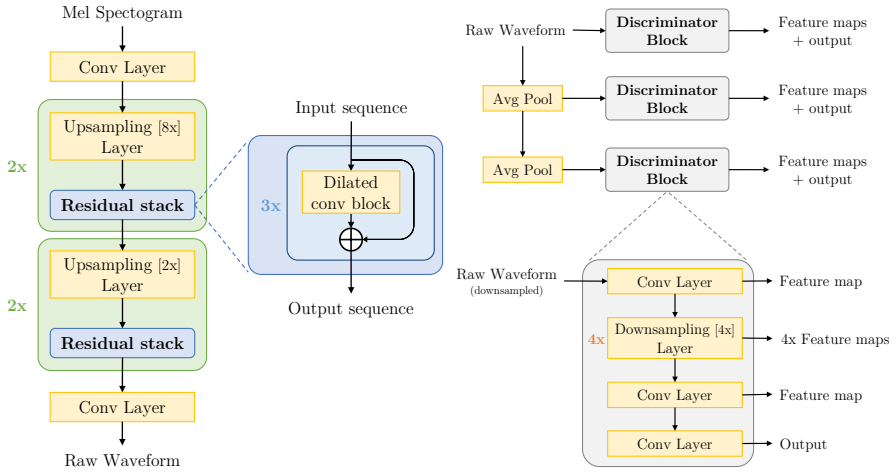
In terms of architecture (reported in Figure 3.16), AutoVC's **encoder** E is composed of a sequence of convolutional blocks (convolution, BN and activation function) and a bi-directional LSTM. The output of the last recurrent module is split into forward and backward features, which are **downsampled** with the same frequency factor to obtain two latent vectors. The concatenation of such vectors (at corresponding indices) is the latent code associated with input speech, while the encoder output is obtained by **upsampling** the latent code to the original time dimension, by copying forward and backward LSTM features in their corresponding direction. AutoVC's **decoder** D is also composed of a sequence of convolutional blocks and LSTM layers (this time uni-directional), followed by a linear projection layer to map the LSTM hidden size to the number of target mel-frequency bins, and a final post-net, acting as a residual [152, 185]. Finally, AutoVC's speaker encoder S is a pre-trained speaker embedding module. Thanks to the use of a pre-trained speaker embedding module, AutoVC is also able to perform the **zero-shot conversion**. In case the zero-shot conversion feature is not required, using one-hot encodings of speaker indices would suffice to perform the traditional many-to-many conversion.

In this thesis, AutoVC is adapted to work with both speaker and dialect accent, by allowing the bottleneck to disentangle both speech properties. Thus, in order to convert speaker identity alone, we fully reproduce the original AutoVC architecture, while for the conversion of dialect accent we explore the possibility of conditioning AutoVC on both speaker and dialect accent embeddings or just the latter.

3.4 Vocoder

Vocoders are modules specialized for waveform generation from a combination of linguistic and acoustic features. Nowadays, vocoders have the goal of spectrogram inversion, i.e. converting linear or mel-scaled spectrograms back to the waveform that originated them, thus acting as a sort of phase reconstruction stage. Given that tasks, terminology and example models used in the vocoding step of a TTS system have already been described in Chapter 2, all that's left is to describe the main vocoder used for spectrogram inversion purposes throughout this thesis, which is done in the following section.

3.4.1 Architectures



(a) Generator. (b) Discriminator.

Figure 3.17: MelGAN architecture [100]. The generator (left) is used to synthesise waveforms as close to ground-truth as possible, in order to fool the discriminator (right).

MelGAN is the main architecture we rely on to convert mel-spectrograms to waveforms. **MelGAN** [100] is the first non-autoregressive feedforward convolutional architecture to perform audio waveform generation in a GAN setup. As opposed to WaveNet, it consists of a generator/discriminator architecture, as shown in Figure 3.17, where the generator's goal is to synthesise waveforms that look as if they were sampled from the underlying data distribution, in order to fool the discriminator.

The **generator** G is a CNN that takes as input a mel-scaled spectrogram and outputs a raw waveform, as shown in Figure 3.17a. Since spectrograms are generated using STFT, which intuitively computes the FFT over fixed-size windows of the input waveform, spectrograms and waveforms have different temporal resolutions, with the former

being defined at a coarser scale. In particular, given a waveform of n samples and STFT parameters f, h, w, which respectively indicate the window length after padding, the hop length between consecutive windows and the window length before padding, the output spectrogram will have the number of frames reported in Equation 3.9, assuming to apply a padding of $\frac{f-h}{2}$ to both sides of the signal.

$$\left\lfloor \frac{2n + f - h - w}{h} \right\rfloor + 1 \tag{3.9}$$

This explains the generator's architectural choice of including **fractionally strided convolutions** to upsample the input signal. Dilated convolutions (with exponentially increasing dilation rates 1, 3 and 9) are instead included to inject the inductive bias of long-range correlation among audio time steps, thanks to their property of exponentially increasing the receptive field with the number of layers.

The **discriminator**, shown in Figure 3.17b, is again a CNN, that adopts a multi-scale architecture by which 3 discriminators (D_1, D_2 and D_3) composed of the same building blocks operate at different temporal resolutions. The first one works at the scale of the generated audio, while the second and the third one downsample the signal by a factor of 2 and 4, respectively. Moreover, authors rely on a windowing strategy, whereby the model learns to classify between distributions of small audio chunks, instead of entire audio sequences.

For what concerns loss functions, the hinge loss version of the GAN objective [109] was used, as reported in Equation 3.10 and Equation 3.11.

$$\begin{aligned} \min_{D_k} \quad & \mathbb{E}_x \left[\min(0, 1 - D_k(x))\right] + \\ & \mathbb{E}_{s,z} \left[\min(0, 1 + D_k(G(s, z)))\right], \forall k = 1, 2, 3 \end{aligned} \tag{3.10}$$

$$\begin{aligned} \min_G \quad & \mathbb{E}_{s,z} \left[\textstyle\sum_{k=1,2,3} -D_k(G(s, z))\right] + \\ & \lambda \textstyle\sum_{k=1,2,3} \mathbb{E}_{x,s\sim p} \left[\sum_{i=1}^{T} \frac{1}{N_i} \|D_k^{(i)}(x) - D_k^{(i)}(G(s))\|_1\right] \end{aligned} \tag{3.11}$$

In Equation 3.10 and Equation 3.11, x represents the raw waveform, s represents the conditioning information (i.e. the mel-spectrogram), z represents the Gaussian noise vector, $D_k^{(i)}$ represents the i-th layer feature map output of the k-th discriminator block, N_i denotes the number of units in each layer and λ is an hyper-parameter ($\lambda = 10$ in the original paper). Also, the second expectation term in Equation 3.11 is a feature matching

objective [102], which minimizes the L_1 distance between the discriminator feature maps of real and synthetic audio.

At the time of writing, MelGAN is not anymore a SOTA model in terms of generated speech quality, as there are other better and more recent alternatives that achieve results closer to ground truth, but the main goal of MelGAN is to make vocoding a negligible part of the whole TTS pipeline, in terms of speed and applicability. Regarding speed, MelGAN can generate speech at $51.9\,\text{kHz}$ on CPU[7] and $2500\,\text{kHz}$[8] on GPU, according to original tests on the same hardware setup[9]. Regarding practical applications, MelGAN can be used as a universal vocoder [114] as it generalizes well to unseen speakers, thus allowing the user to skip tedious fine-tuning steps. Thus, we decide to rely on a pre-trained version of MelGAN for such conveniences.

3.5 End-to-end approach

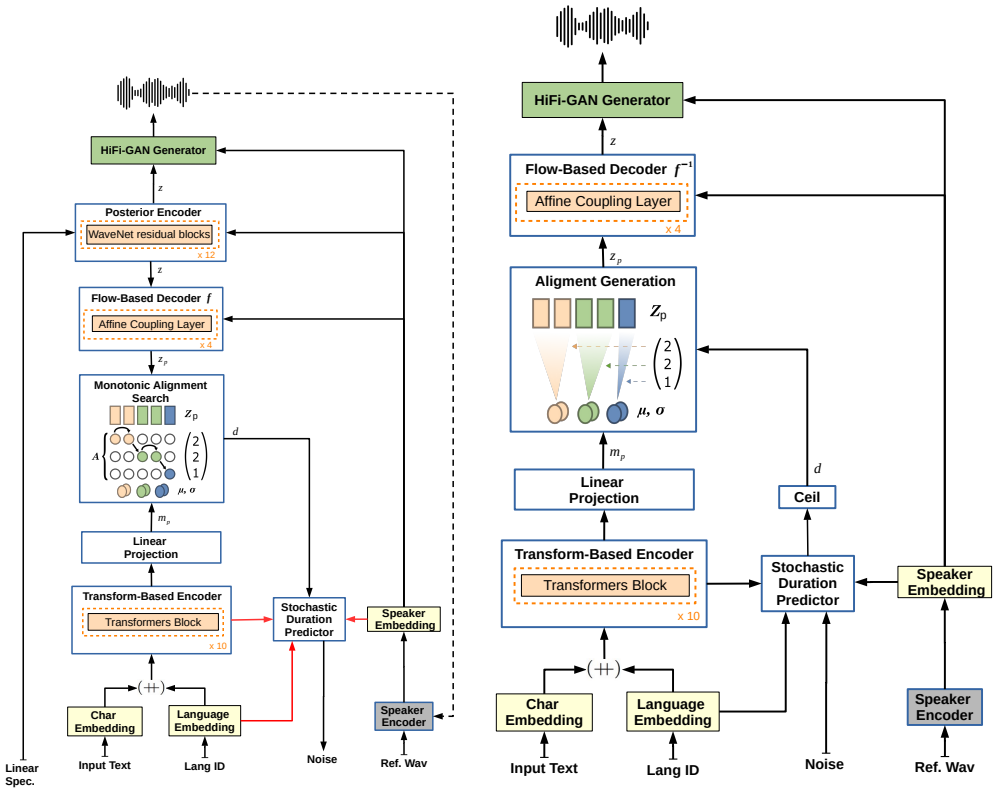
In this section, we introduce an end-to-end model that incorporates multi-lingual synthesis and zero-shot multi-speaker TTS, that will be used as a further comparison to the pipelines described so far. The model we are referring to is **YourTTS** [20], a SOTA architecture which can synthesize voices in multiple languages and reduce data requirements significantly, by transferring knowledge among languages in the training set. For example, in the paper, Casanova et al. show that they can easily introduce Brazilian Portuguese to the model with a single speaker dataset by co-training with a larger English dataset, thus enabling the possibility of making the model speak Brazilian Portuguese with voices from the English dataset.

YourTTS is an extension of previous work from the same authors, i.e. SC-GlowTTS [19], which relies on the Variational Inference with adversarial learning for end-to-end Text-to-Speech (VITS) model [83] as the backbone architecture and builds on top of it. Differently from VITS, YourTTS uses a larger text encoder and employs a separately trained speaker encoder model, namely H/ASP [67], to compute speaker embedding vectors. The speaker encoder was trained with a combination of prototypical angular [29] and CE loss functions on the VoxCeleb2 [28] dataset. Additionally, YourTTS relies on raw graphemes as input text, in order to support low-resource languages. Moreover, for

[7] Tested on only 1 CPU core.

[8] N kHz means that the model can generate $N \times 1000$ raw audio samples per second.

[9] Authors relied on an NVIDIA GTX 1080 Ti GPU and an Intel® Core™ i9-7920X CPU @ 2.90 GHz processor.



(a) Training procedure. (b) Inference procedure.

Figure 3.18: YourTTS architecture at training and inference time [20].

multi-lingual training, a 4-dimensional trainable language embedding is concatenated to each input character.

In terms of architecture, VITS relies on a mix of **specialized optimizations**, such as adversarial learning, and modern DL modules, such as normalizing flows [91], VAEs and Transformers [177] to achieve high-quality natural-sounding output. VITS improves on the model it's based on, i.e. GlowTTS [82], by replacing the duration predictor with a stochastic duration predictor, so as to better model variability in speech, and by connecting a HiFi-GAN vocoder [94] to the decoder's output through a VAE, so as to let the model learn an intermediate representation different from traditional mel-spectrograms and enable end-to-end training. A thorough explanation of the VITS architecture is considered out of scope for this thesis, but we encourage the reader to refer to [83] for further details.

YourTTS during training and inference is illustrated in Figure 3.18, where $+\!\!+$ indicates concatenation, red connections mean that no gradient will be propagated through them, and dashed connections are optional. Also, the Hifi-GAN discriminator network is omitted for simplicity.

Hence, YourTTS is a powerful architecture that solves many problems in the TTS field, such as multi-lingual TTS (i.e. a single model for multiple languages), multi-speaker TTS (i.e. a single model for multiple speakers), zero-shot learning (i.e. the ability to synthesise speech with unseen speakers), speaker and language adaptation (i.e. the ability to fine-tune pre-trained models to learn new speakers and languages), cross-language voice transfer (i.e. transferring a voice from its original language to a different one) and more. In this thesis, we rely on a pre-trained version of YourTTS, publicly available through Coqui's TTS repository[10], and rely on transfer learning to adapt it to the task of **multi-dialect synthesis**.

[10] https://github.com/coqui-ai/TTS

CHAPTER 4

DATASET

This chapter describes all the details about data sources and data transformations that were used for experimentation. Section 4.1 presents information about data composition for each considered corpora, while Section 4.2 gives a detailed overview of all the pre-processing steps applied to both text and audio. This thesis relies on single-speaker and multi-speaker datasets to assess implementation correctness for both acoustic and voice conversion models, while it makes use of multi-speaker multi-dialect corpora to test the solutions described in Chapter 3.

4.1 Data resources

The speech domain (and the TTS one in particular) suffers from a lack of high-quality open-source data. Recent projects, such as Common Voice [4], have greatly simplified the work of many in the ASR field, given its massive scale[1], but most large-scale corpora (including Common Voice), do not meet the high bar required to build effective TTS systems. Indeed, in order to obtain a human-like voice in the TTS setting, audio files usually need to be studio-recorded by professional voice talent and audio needs to be carefully cleaned, with post-processing steps such as prooflistening[2].

[1] As of the end of July 2021, version 7.0 of the Common Voice corpus contains more than 11 k hours of validated speech in 76 languages.

[2] Prooflistening is a process whereby an audio expert listens to recordings and looks for errors and inaccuracies, such as text/audio mismatch, background noise, unclear articulation, too short or too long pauses and so on.

Some datasets containing such high-quality speech do exist in the public domain, but most of them fall short on scale. Nevertheless, the following sections introduce the set of corpora selected for this thesis. All datasets were analysed under the same settings; in particular, Table 4.1 shows definitions used to report transcriptions statistics throughout the whole chapter.

	Definition
Characters	A character is any symbol in the input text
Words	A word is an entry of a whitespace-tokenized text
Letters	A letter is a character in $\{[a-z], [A-Z], [0-9]\}$
Punctuation	A punctuation mark is a character in *!"#$%&\'()*+,-./:;<=>?@[\\]^_‘{|}~*

Table 4.1: Transcriptions statistics definitions, that clearly state what is meant by characters, words, letters and punctuation marks.

4.1.1 LJ Speech

The LJ Speech corpus [75] is a public domain speech dataset consisting of 13 100 short audio clips of a **single American English female speaker** reading passages from 7 non-fiction books, together with corresponding transcriptions. Both text and audio are in the public domain, with the former published between 1884 and 1964 and the latter recorded in 2016-17 by the LibriVox project [108]. Each audio file is a single-channel 16-bit Pulse Code Modulation (PCM) Waveform Audio File Format (WAV) file with a sample rate of 22 050 Hz, that have been segmented automatically based on silences in the original full-length recordings (clip boundaries generally align with sentence or clause boundaries). The text was then matched to the audio manually, and a Quality Assurance (QA) pass was done to ensure that the text accurately matched the words spoken in the audio.

As reported in LJ Speech's website [75], transcriptions with numbers, ordinals, and monetary units are already expanded into full words, even though certain abbreviations (e.g. *Mr.*, *Dr.*, *No.*) are left as-is and have to go through additional processing. Other steps are also needed to normalize text, as further analysis of transcriptions shows that the following non-standard characters, other than unconventional quotes, are included in certain utterances: *"à, â, è, é, ê, ü"*. LJ Speech's website points out that exactly 19 of the transcriptions contain non-ASCII characters (e.g. *"raison d'être"*).

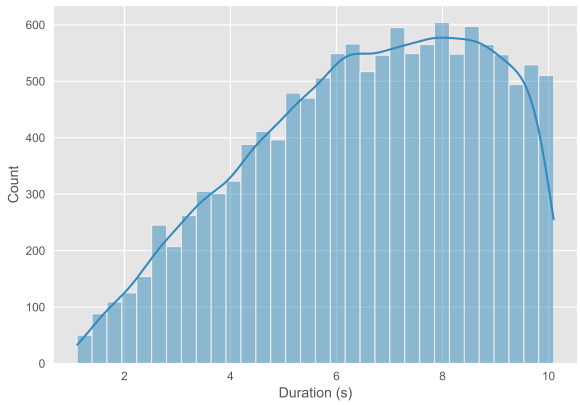


Figure 4.1: Histogram of audio durations (in seconds) for LJ Speech recordings.

Table 4.2 shows statistics related to text transcriptions in the LJ Speech dataset, while Figure 4.1 shows the distribution of audio snippet durations (in seconds), with the minimum duration being $1.11\,\mathrm{s}$, the maximum $10.10\,\mathrm{s}$, the average $6.57 \pm 2.19\,\mathrm{s}$ and the total one $23.92\,\mathrm{h}$.

In this thesis, version 1.1 of LJ Speech was used.

	Min	**Max**	**Mean**	**Std**	**Unique**
Characters	5	187	98.35	34.05	88
Words	1	44	19.02	6.68	14840
Letters	4	155	80.09	27.86	62
Punctuation	0	19	2.37	1.62	15

Table 4.2: LJ Speech transcriptions statistics, computed over all ground-truth texts.

4.1.2 LibriSpeech

As LJ Speech, the LibriSpeech corpus [134] is a dataset of read American English speech derived from audiobooks that are part of the LibriVox project [108]. Differently from LJ Speech, it is a **multi-speaker dataset** that acts as a large scale successor of the volunteer-supported speech-gathering effort VoxForge [179], which has around 100 hours of English speech and suffers from major gender and per-speaker duration imbalances.

Table 4.3 shows the various LibriSpeech data partitions freely available. In this thesis, version *train-clean-100* of LibriSpeech was used.

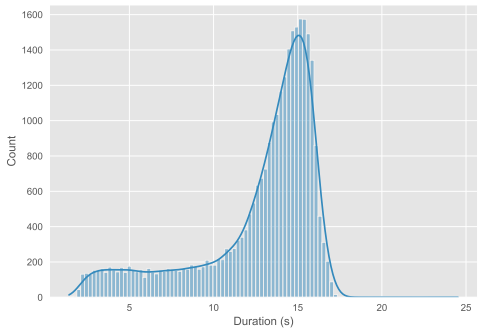
Further analysis of the *train-clean-100* partition shows that text transcriptions are

Subset	Hours	Per-speaker minutes	Female speakers	Male speakers	Total speakers
dev-clean	5.4	8	20	20	40
test-clean	5.4	8	20	20	40
dev-other	5.3	10	16	17	33
test-other	5.1	10	17	16	33
train-clean-100	100.6	25	125	126	251
train-clean-360	363.6	25	439	482	921
train-clean-500	496.7	30	564	602	1166

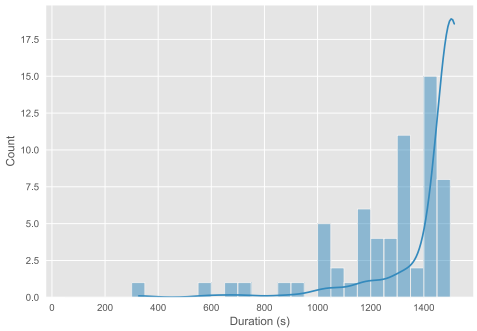
Table 4.3: Data subsets in Librispeech. The difference between *clean* and *other* is the quality of the audio and its corresponding transcription. The *clean* quality is higher than the *other*.

clean, as there are no non-ASCII characters in any of the $28\,539$ prompts. Moreover, the split has a total amount of 251 English speakers, with each speaker having an average of 113.70 ± 15.18 utterances and an average total duration of $1442.74\,\mathrm{s} \pm 161.29\,\mathrm{s}$. Thus, the majority of speakers have only around $20\,\mathrm{m}$ of audio or less.

Table 4.4 shows statistics related to text transcriptions in the *train-clean-100* partition of LibriSpeech dataset, while Figure 4.2 shows the distribution of audio snippet durations (in seconds) out of all speakers, with the minimum duration being $1.41\,\mathrm{s}$, the maximum $24.53\,\mathrm{s}$, the average $12.69\,\mathrm{s} \pm 3.57\,\mathrm{s}$ and the total one $100.59\,\mathrm{h}$.



(a) Distribution of audio durations over all recordings.



(b) Distribution of per-speaker audio durations.

Figure 4.2: Histograms of audio durations (in seconds) for LibriSpeech recordings in the *train-clean-100* partition.

	Min	Max	Mean	Std	Unique
Characters	8	398	184.65	58.65	28
Words	2	76	35.03	11.43	32456
Letters	7	331	150.62	47.77	26
Punctuation	0	8	0.34	0.76	1

Table 4.4: LibriSpeech transcriptions statistics, computed over all ground-truth texts in the *train-clean-100* partition.

4.1.3 VCTK

The VCTK [194] corpus contains speech data uttered by 110 **English speakers with various accents**, along with corresponding transcriptions. Each speaker reads about 400 sentences from the Herald Glasgow newspaper, selected so as to increase phonetic coverage [178]. Some passages are the same for all speakers, such as the rainbow passage[3] and the elicitation paragraph[4].

All speech data was recorded in a professional studio at the University of Edinburgh, with a sample rate of 96 kHz and quantization of 24 bits. All recordings were then converted into 16 bits and downsampled to 48 kHz.

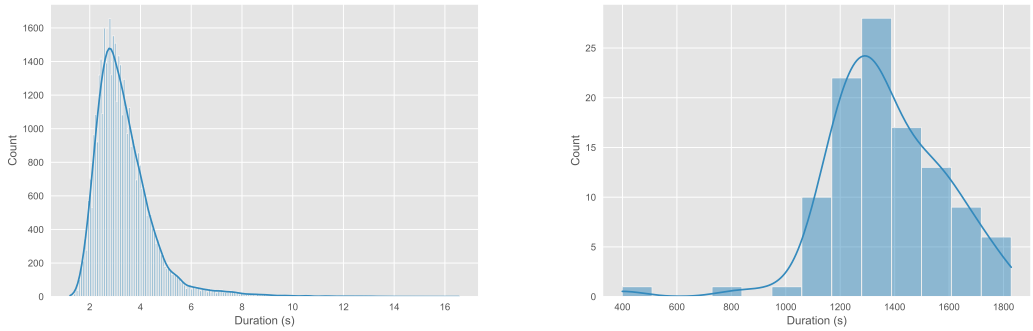
Also, some prompts are excluded from the dataset because of technical issues. In particular, all audio recordings from speakers *p280* and *p315* are removed and some recordings from *p362* are also deleted, because of the lack of either text or audio files. Further analysis of text transcriptions shows that some of the 43 873 utterances contain non-standard characters, such as newlines and tabs.

This corpus was originally aimed for HMM-based TTS systems, but is also suitable for DNN-based multi-speaker TTS and waveform modelling, as reported by the authors in the official dataset website[5].

Table 4.5 shows statistics related to text transcriptions in the VCTK dataset, while Figure 4.3 shows the distribution of audio snippet durations (in seconds) out of all speakers, with the minimum duration being 1.22 s, the maximum 16.56 s, the average 3.37 s ± 1.20 s and the total one 41.04 h.

The dataset has a total amount of 108 English speakers, with each speaker having an average of 406.23 ± 50.34 utterances and an average total duration of 1367.94 s ±

[3] http://web.ku.edu/~idea/readings/rainbow.htm
[4] http://accent.gmu.edu
[5] https://datashare.ed.ac.uk/handle/10283/3443



(a) Distribution of audio durations over all recordings.

(b) Distribution of per-speaker audio durations.

Figure 4.3: Histograms of audio durations (in seconds) for all VCTK recordings.

210.54 s. Thus, as in LibriSpeech, the majority of speakers have only around 20 m of audio or less.

	Min	Max	Mean	Std	Unique
Characters	10	186	39.95	19.83	62
Words	3	40	8.73	3.78	5776
Letters	6	146	31.18	16.19	52
Punctuation	0	6	1.35	0.62	7

Table 4.5: VCTK transcriptions statistics, computed over all ground-truth texts and for all speakers.

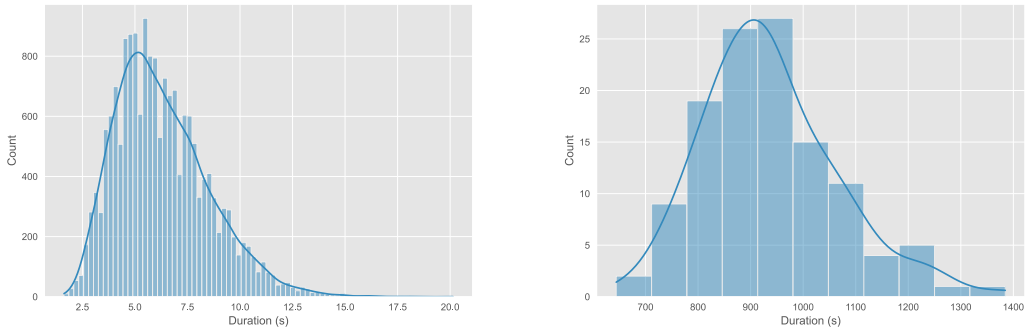
4.1.4 SLR83

SLR83 [42] is a dataset that contains male and female transcribed high-quality audio of **British English from various dialects** of the UK and Ireland. The dialects represented in the recordings are Irish English, Midlands English, Northern English, Scottish English, Southern English and Welsh English.

The dataset stems from a **crowdsourcing initiative**, with volunteers above 21 years of age. The participants were both Google employees in London (101 in total) and students, friends and family of collaborators at the University of Cardiff (the remaining 19 speakers).

Different **elicitation scripts** were generated for each speaker, however, a few sentences were included in all the elicitation scripts. The scripts were crafted for each dialect so that specific linguistic features present in that dialect could be contrasted against other dialects, as well as capturing pronunciation variants of, for example, place names. Line sources vary widely and include data gathered from Wikipedia [189], the rainbow passage (the same one used in VCTK), Alice's Adventures in Wonderland [16], lines that are intended to be spoken by a virtual assistant and more. The following is the list of elicitation line categories included in the dataset.

- Idiolect: lines included to differentiate the speaker-specific aspects of the accent of a speaker from the regional accent (same lines for every speaker);
- Global English accents: lines included to reflect the phonological aspects of a wide range of global English accents; they include, among other words, the names of the most populated cities of the world, major airlines, the most popular US cities, and popular English personal names;
- British Isles: lines included to differentiate between global English accents and all British Isles accents;
- Accent region: lines included to differentiate regional accents of the British Isles among each other.



(a) Distribution of audio durations over all recordings.

(b) Distribution of per-speaker audio durations.

Figure 4.4: Histograms of audio durations (in seconds) for all SLR83 recordings.

The corpora contains a total of 17 877 recordings, stored at 48 kHz with a depth of 16 bits per sample. Table 4.7 shows statistics related to text transcriptions in the SLR83

dataset, while Figure 4.4 shows the distribution of audio snippet durations (in seconds) out of all speakers, with the minimum duration being $1.62\,\mathrm{s}$, the maximum $20.14\,\mathrm{s}$, the average $6.30\,\mathrm{s} \pm 2.23\,\mathrm{s}$ and the total one $31.23\,\mathrm{h}$ (around 10 hours less than VCTK). Instead, audio durations broken down by dialect are shown in Figure 4.5 and audio statistics are summarized and presented for dialects in Table 4.6.

The dataset has a total amount of 120 British English speakers (49 female and 71 male), with each speaker having an average of 148.69 ± 5.58 utterances and an average total duration of $936.92\,\mathrm{s} \pm 128.49\,\mathrm{s}$. Thus, this corpora exhibits a slightly lower average speaker coverage than LibriSpeech and VCTK, with many speakers having only around $15\,\mathrm{m}$ of audio or less.

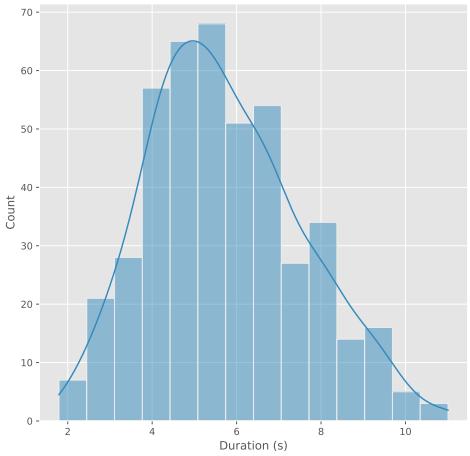
Finally, further analysis of transcriptions shows that the following non-standard characters, other than anomalous dashes and quotes, are included in certain utterances: *"á, è, é, ô, ö, ü, ă"*, which means that further text processing is required, as observed in the LJ Speech dataset.

	Count	**Min**	**Max**	**Mean**	**Std**
Utterance	17843	1.62	20.14	6.30	2.23
Speaker	120	644.27 (*mif_02484*)	1384.62 (*wef_12484*)	936.92	128.49
Dialect	6	2576.21 (*irish*)	19376.55 (*welsh*)	18738.30	16452.43

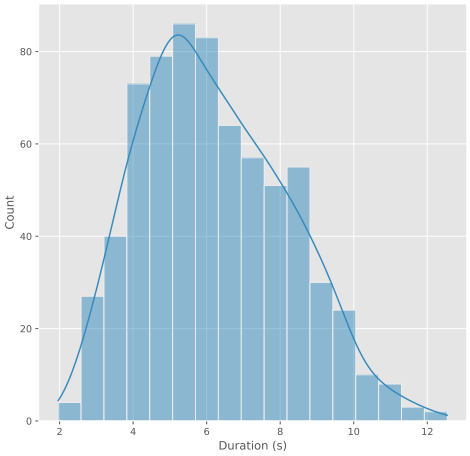
Table 4.6: Statistics of audio durations (in seconds) broken down at the utterance, speaker and dialect level, for all SLR83 recordings.

	Min	**Max**	**Mean**	**Std**	**Unique**
Characters	9	176	79.08	32.02	76
Words	2	28	13.83	5.35	7567
Letters	8	151	66.19	27.09	62
Punctuation	0	6	0.18	0.49	2

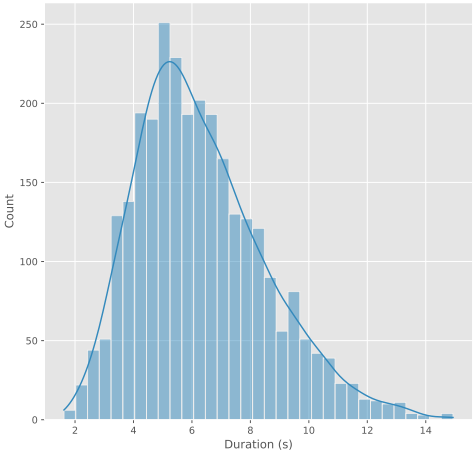
Table 4.7: SLR83 transcriptions statistics, computed over all ground-truth texts and for all speakers and dialects.



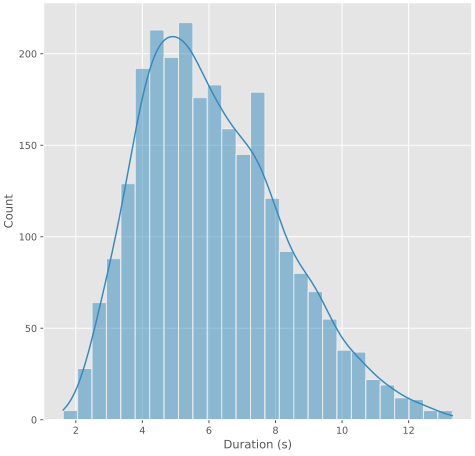
(a) Irish speakers audio durations.



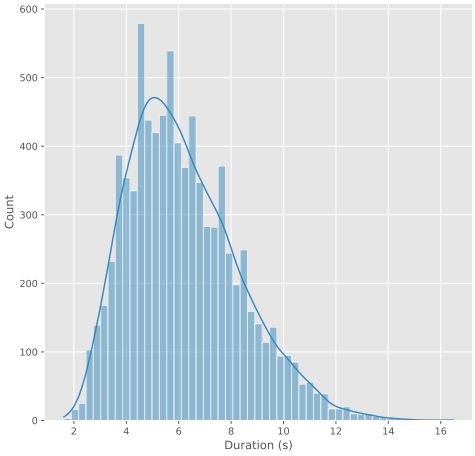
(b) Midlands speakers audio durations.



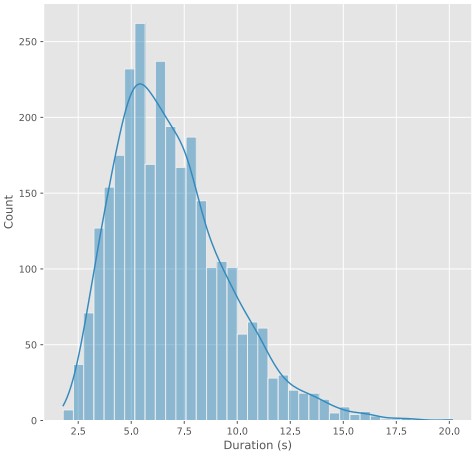
(c) Northern speakers audio durations.



(d) Scottish speakers audio durations.



(e) Southern speakers audio durations.



(f) Welsh speakers audio durations.

Figure 4.5: Histograms of audio durations (in seconds) for all SLR83 recordings, broken down by dialect accent.

4.1.5 Others

There are also other datasets that provide information about the dialect accent of its speakers. Some of them are in the public domain, such as Intonational Variation in English (IViE) [170] and Corpora e Lessici dell'Italiano Parlato e Scritto (CLIPS) [33], while some others have more restrictive licenses, such as Accents of the British Isles (ABI) (both ABI-1 [169] and ABI-2 [1]).

The IViE corpus contains recordings of nine urban dialects of English spoken in the British Isles, as shown in Figure 4.6. Recordings of male and female speakers were made in London, Cambridge, Cardiff, Liverpool, Bradford, Leeds, Newcastle, Belfast and Dublin. Also, three speakers are from the ethnic minorities of bilingual Punjabi/English, bilingual Welsh/English and Caribbean descent. This dataset was originally built for research in linguistics and then adopted for ASR modelling, but its relatively low quality in terms of audio recordings and the lack of the majority of text transcriptions makes it unusable for TTS purposes. In particular, in its first version, only 4 hours out of the 36 total hours were labelled.

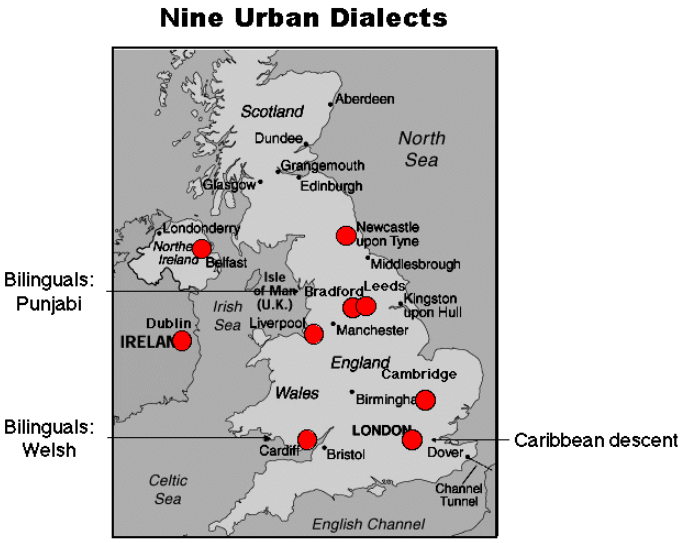


Figure 4.6: Dialects in the IViE corpus [170]. The red dots roughly indicate the location of native speakers in the dataset.

Next, the ABI-1 corpus is a collection of speech recordings of 280 speakers sampled throughout the British Isles, with more than 70 hours of recordings covering 13 distinct accent regions. For each region, an average of 10 male and 10 female subjects each recorded around 15 minutes of read speech. Also, speakers' ages ranged from 16 to 79.

Instead, the ABI-2 corpus contains 286 speakers covering 13 accent regions of the British Isles which are not covered in the original ABI-1 corpus. Statistics are similar to ABI-1, with ABI-2 also having 70 hours of recordings. Thus, the ABI family covers together 26 accent regions in the United Kingdom (as shown in Table 4.8), with a total of 140 hours of speech and 566 speakers. Unfortunately, neither of the two is open source, not even for academic purposes.

ABI-1	**ABI-2**
Belfast	Bristol
Birmingham	Caernarfon
Burnley	Cardiff
Denbigh	Coalville
Dublin	Dudley
Elgin	Edinburgh
Glasgow	Hartlepool
Hull	Hereford
Liverpool	Leeds
Lowestoft	Shrewsbury
Newcastle	Southend-on-Sea
Tower Hamlets	Stoke-on-Trent
Truro	Yeovil

Table 4.8: Dialect accent regions in the ABI corpora. The left column indicates the location of native speakers in the ABI-1 dataset, while the second column does the same for speakers in ABI-2.

Another dataset that contains information on British dialect accents at the regional level is FReiburg corpus of English Dialects (FRED) [55]. Such corpus was compiled by the University of Freiburg during the period 2000 through 2005 and consists of 370 unique prompts, which total around 2.5 million words of text and 300 hours of speech (excluding interviewer utterances), covering nine major dialect areas and a multitude of locations, as shown in Figure 4.7. Even though the dataset was compiled starting from the year 2000, recordings were collected between 1968 and 2000, with the majority of them (40.5%) in the period ranging from 1980 to 1989. Also, speakers in the corpus were born between 1877 and 1959, with 89% having their birthdays before 1920. Due to the relatively low quality of audio recordings and the fact that such old recordings may not represent the modern English language anymore, FRED is not considered as

a suitable candidate to build robust TTS systems. Moreover, the full FRED corpus is only available to researchers and (visiting) scholars at the University of Freiburg, due to copyright restrictions. Only a 123 hours subset with 5 dialect varieties (denoted FRED-S) is available in the public domain.

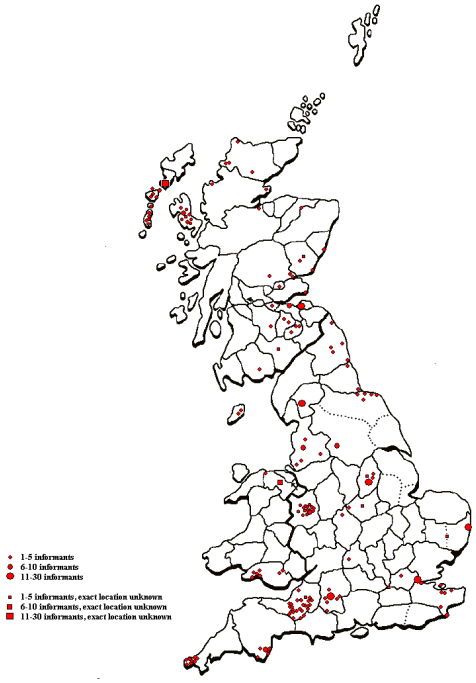


Figure 4.7: Location of informants in the FRED corpus [55]. The red dots roughly indicate the location of native speakers in the dataset.

Finally, the last dataset taken into consideration for dialect accent modelling is CLIPS [33]. This corpus differs from all the others as it doesn't contain British English data, but spoken Italian. CLIPS amounts to a total of 100 hours of transcribed speech, uniformly balanced in terms of speaker gender, from 15 Italian cities: Bari, Bergamo, Bologna, Cagliari, Catanzaro, Firenze, Genova, Lecce, Milano, Napoli, Palermo, Parma, Perugia, Roma and Venezia. The corpus is sub-divided into 4 categories: radio and television (e.g. news, interviews, talk shows), on-the-field collection (with recordings on map tasks and spot-the-difference games), read text and phone-recorded speech. Out of the 4, only the read text section sounds clean enough to be used for a TTS system, while the other recordings suffer from long pauses, usage of too many filler words and people talking over each other, especially in interviews.

Thus, all the corpora presented in this section had to be excluded for further processing because of either license issues, scale or quality or a combination of them, leaving SLR83 as the best option to model dialect accents, as introduced in Chapter 3.

4.2 Data processing

As already hinted throughout the thesis, TTS systems have the need to work with clean data. This usually means that both audio recordings and the corresponding text transcriptions need to go through a sequence of pre-processing steps. Some steps are always applied, while some others might depend on the downstream task, or may act as a form of data augmentation and model regularization, thus becoming optional stages in the overall data processing flow. Subsection 4.2.1 and Subsection 4.2.2 describe all the data transformations, used for experimentation, that were applied to raw input data.

4.2.1 Text

As introduced in Section 2.3, raw input text needs to be converted to a form more suitable for further linguistic/acoustic processing. In this thesis, raw text is first lower-cased and **verbalized** using the NeMo [99] text normalization tools [198, 199], which rely on weighted finite-state transducers. Then, the normalized text is **tokenized** into words by splitting on whitespaces, as the only considered language is English, and the last remaining step is to convert text to a list of indices. An optional step right after normalization would be to convert **graphemes to phonemes**. In this thesis, we mainly rely on character-based inputs, even though an implementation of a G2P module, based on the phonemizer [10] open-source tool, is provided[6].

Figure 4.8 shows an example of a text transcription converted to integer indices, where dashed arrows indicate optional paths, i.e. the user can decide whether or not to transduce graphemes to phonemes.

Table 4.9 shows the input-to-grapheme and grapheme-to-input mappings, along with the explanation of certain tokens that convey a special meaning. For example, the "_" token is used to right-pad text indices to the maximum length in a batch, "^" and "~" are used to mark the start and end of sentences, in order for the acoustic model to make sense of initial and final pauses in recordings, "+" marks the end of a word and the beginning of another one and it might be useful for the model to learn breaks within the utterance, while "@" is a fail-safe bucket where all tokens unseen during training fall into.

[6]Phonemization is based on the eSpeak backend and applied with the *en-GB* locale, with preserved punctuation marks and without stresses.

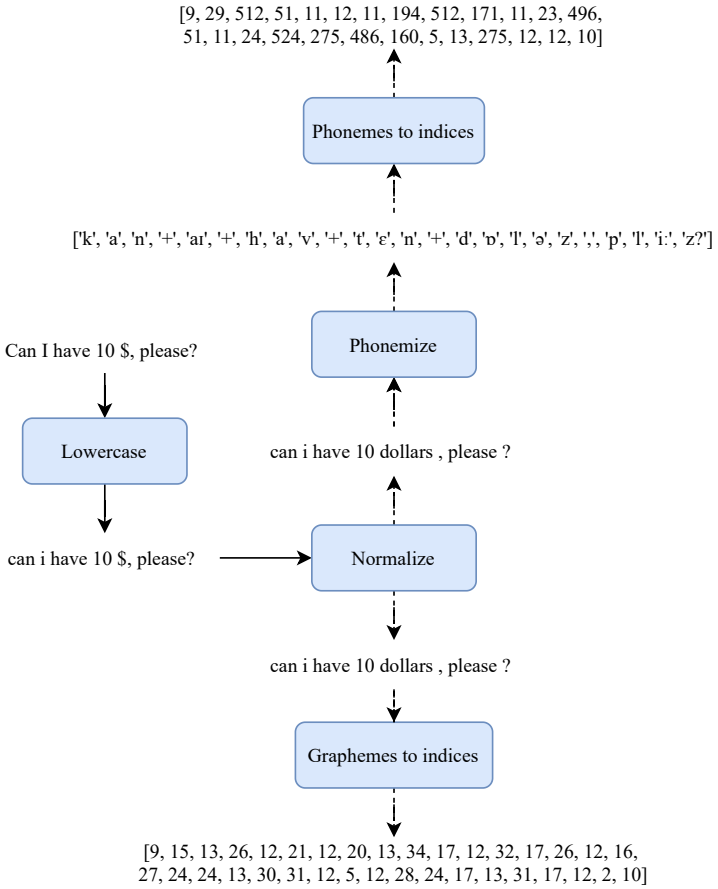


Figure 4.8: Example of raw text to indices pre-processing. The input text is lower-cased, normalized and possibly phonemized before being mapped to a set of integers.

For what concerns the phoneme-to-index and index-to-phoneme mapping, the **IPA**[7] standard is followed, which is one of the most popular alphabetic systems of phonetic notation.

4.2.2 Audio

Audio pre-processing is the most important and laborious stage of the data processing workflow, as there are many moving parts and hyper-parameters to tweak. First of all, there's the need to differentiate between transforms that happen in the time domain and the ones that need to be applied in the frequency domain. From now on, the time-domain representation is going to be a raw waveform, while the frequency domain representation is a spectrogram.

Given that this thesis deals with multiple independent modules, each one of them

[7] Some well-estabilished implementations of IPA include SAMPA and X-SAMPA, with the former only covering part of the IPA symbols and latter being a full encoding of IPA.

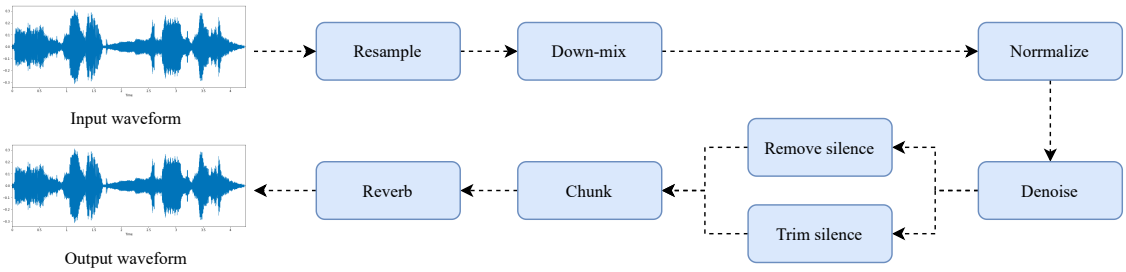
Index	Grapheme	Meaning
0	_	Padding
1	*!*	-
2	*?*	-
3	*"*	-
4	*'*	-
5	*,*	-
6	*.*	-
7	*;*	-
8	*:*	-
9	^	Start of sentence
10	~	End of sentence
11	+	Word boundary
12	@	Unknown
13-38	*a-z*	-

Table 4.9: Static mapping between graphemes and integer indices.

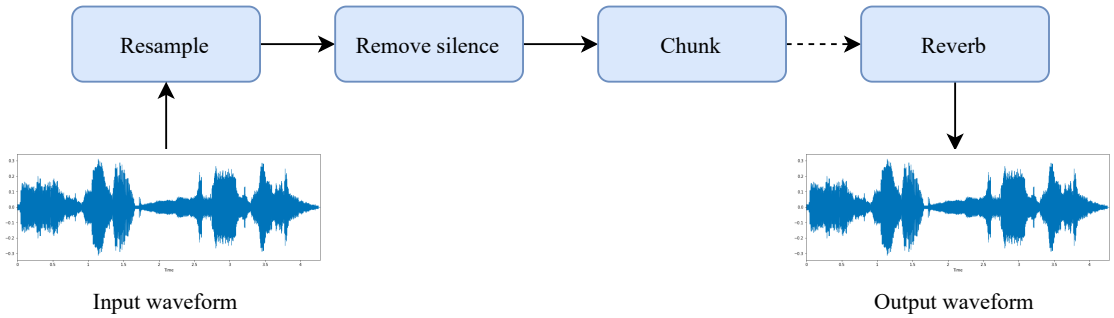
relies on a slightly different waveform and spectrogram pre-processing workflow. Figure 4.9 depicts such differences. In particular, Figure 4.9a shows the overall waveform flow, which is adapted for audio embedding and audio conversion modules in Figure 4.9b and for TTS modules in Figure 4.9c. Instead, Figure 4.9d shows the spectrogram flow shared by all modules. In the figures, bold lines indicate mandatory steps, while dashed lines represent optional steps. Each module sketched in Figure 4.9 will be thoroughly described in the following sections.

Time-domain pre-processing

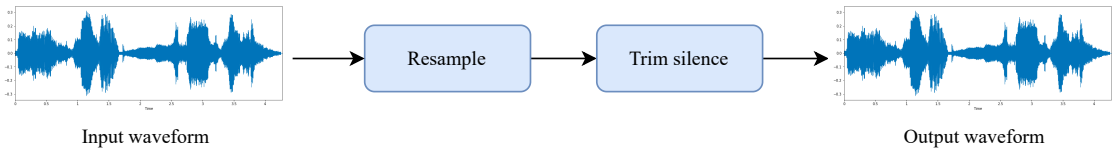
To start off, a raw input waveform needs to be loaded, **resampled** to the target sample rate (e.g. down-sampled from $48\,000$ Hz to $22\,050$ Hz), possibly **down-mixed** to mono (in case the audio signal is stereo) and possibly **normalized** to a standard range such as $[0, 1]$ or $[-1, 1]$. The down-mixing step is usually mandatory, as all subsequent processing assumes to work with single-channel waveforms, while normalization is optional and dependent on the downstream task. According to Librosa [119], which is the reference library for audio and music signal analysis in Python, an input waveform w can be normalized as in Equation 4.1, where c is usually set to $\mu_w = \frac{1}{n}\sum_i^n w_i$ and $\|w\|_p$ is



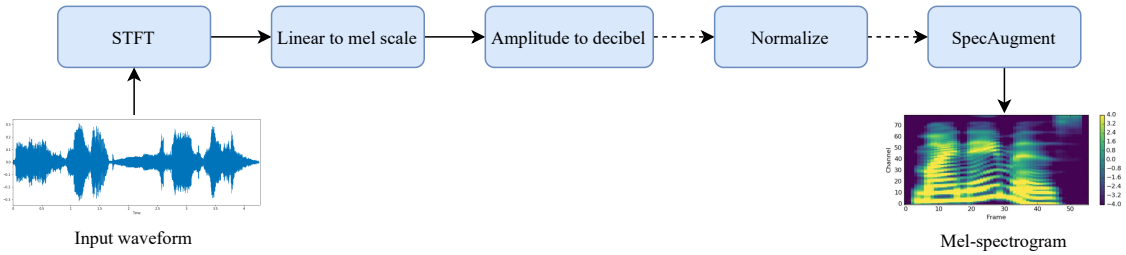
(a) Complete waveforms flow, from the raw waveform to its fully processed version.



(b) Audio embedding and audio conversion modules waveforms flow, in which silence removal and chunking are key steps.



(c) TTS modules waveforms flow, where only resampling and silence trimming are mandatory steps.



(d) Spectrograms flow, from raw waveforms to normalized and augmented mel-spectrograms.

Figure 4.9: Audio pre-processing steps. The complete waveform flow is adapted to each type of module, while the spectrogram flow is common to all of them.

the p-norm of vector w, with the most common type of norm being the infinite one, i.e. $\|w\|_\infty = \max w$.

$$\frac{w - c}{\|w\|_p} \tag{4.1}$$

Other optional steps may be carried out in the time domain, as they might enhance the robustness or generalization abilities of the to-be-trained models. As described in Section 4.1, many open-source speech datasets lack high-quality recordings, as most available speech corpora are volunteer-based; thus, recordings could present some form of background noise (such as room reverberation, hand-clapping, people talking and the like) that is detrimental for many speech systems, such as TTS ones. To get rid of background noise, **speech enhancement** models can be used. One of the most effective and easy-to-use models of such is denoiser [39], as it runs in real-time on a laptop CPU and is capable of removing various kinds of background noise including stationary and non-stationary noises, as well as room reverb, as suggested by empirical evidence. Figure 4.10 shows denoiser's encoder-decoder architecture with skip-connections. For the purpose of this thesis, the pre-trained[8] version of denoiser is used.

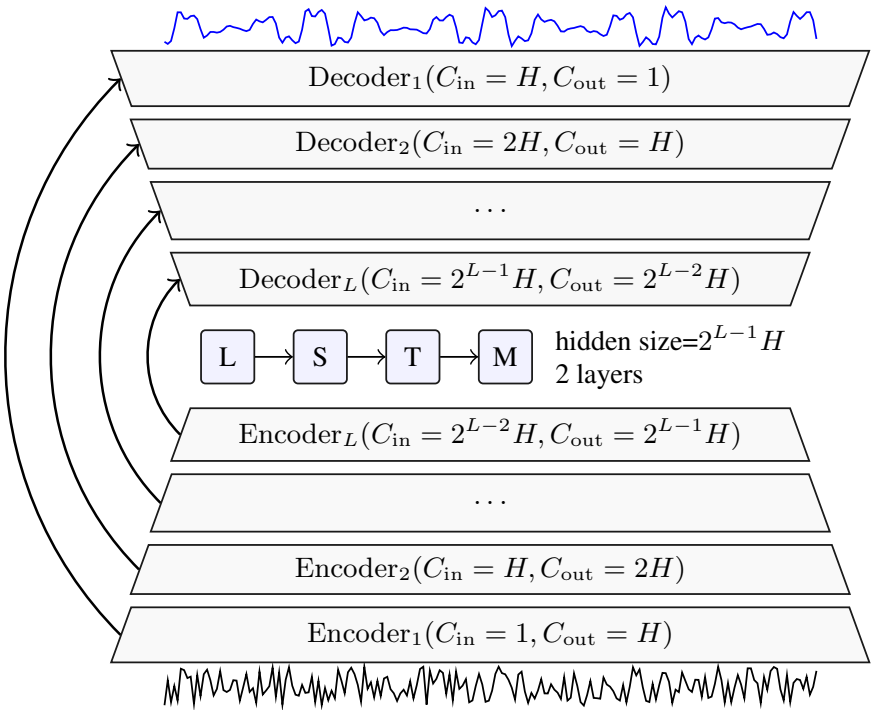
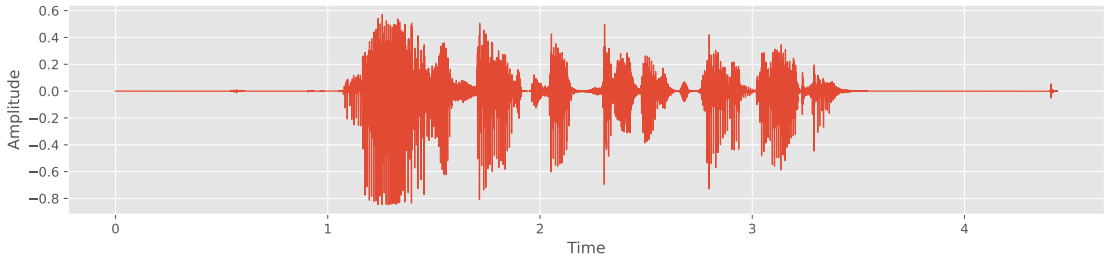


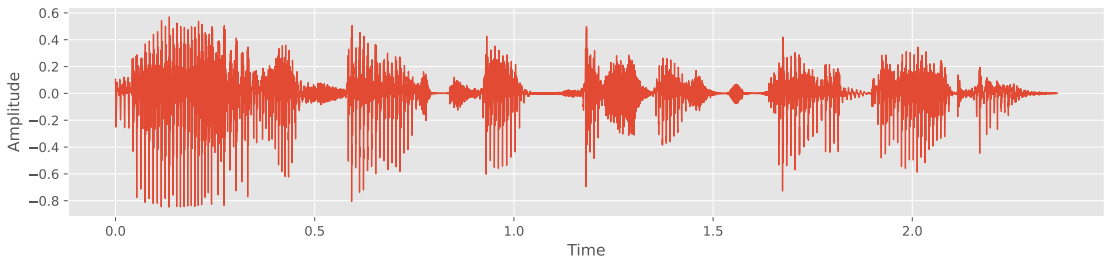
Figure 4.10: Facebook's denoiser architecture [39], composed of an encoder-decoder structure with skip connections.

[8] https://github.com/facebookresearch/denoiser

After denoising, one might want to remove silence chunks from the waveform. This can be useful for TTS systems, as acoustic models would have a hard time trying to align text with spectrogram frames corresponding to silence (unless special tokens are used, such as start and end of sentence ones); in the case of TTS, empirical evidence suggests to only trim silence at the beginning and end of waveforms, as breaks in-between sentences may be helpful for the model to learn an effective pausing strategy. Other than TTS, silence removal can also be useful for tasks such as speaker/dialect verification, given that in such applications audio is usually chunked into small pieces that are classified individually and then aggregated to provide a single output; it's clear that in this case silences need to be removed from the entire audio (at the beginning, in the middle and at the end), as classifying a silence region would simply result in a random output. Figure 4.11 shows an example of removing silence regions for the raw input waveform depicted in Figure 4.11a.



(a) Raw waveform.



(b) Waveform with silence regions removed.

Figure 4.11: Silence removal through Voice Activity Detection (VAD). Notice how the low-amplitude regions at the beginning and end of the original waveform (above) are removed from the output of the VAD model (below).

In this thesis, **silence removal** is handled through VAD models, and in particular using the pre-trained enterprise-grade Silero VAD checkpoints [168], given their efficiency (one audio chunk of 30 ms takes around 1 ms to be processed on a single CPU thread) and accuracy (as shown in Figure 4.12). Results in Figure 4.12 are computed on a closed-source test-set compiled by the Silero team, with 30+ languages, 2200 utterances with

an average duration of 7 seconds, coming from vastly different audio domains (calls, user recordings and various audios sourced from the Internet). Something to keep in mind when using Silero VAD is that, at the time of writing, only pre-trained models are available, with no paper, dataset and code made available to the open-source community.

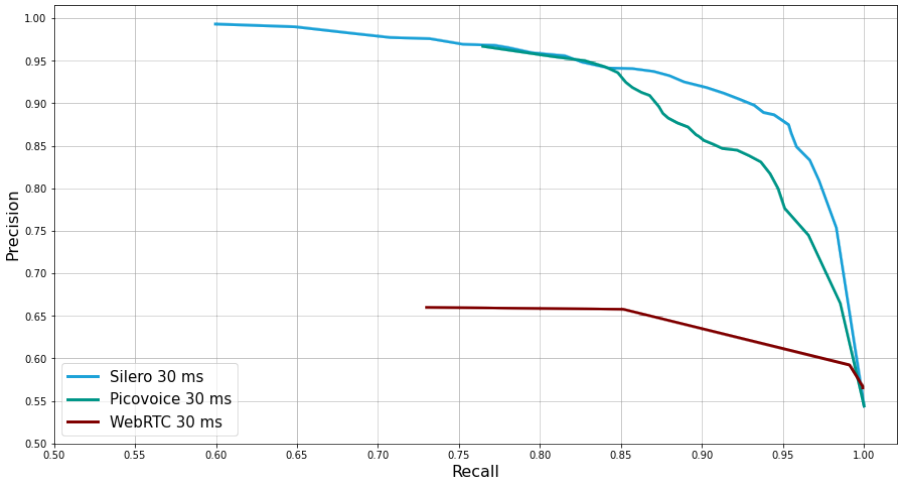


Figure 4.12: Silero VAD [168] test-set precision-recall curve and comparison with other VAD models. Notice how Silero VAD is much closer to the *utopia* point than competing models.

Next, as introduced above some applications might need to work with chunks of audio, instead of the full-length waveform, or might rely on this pre-processing step for regularization purposes. Thus, given a full length input waveform $w_{1,\dots,n}$ (s seconds long), **random chunks** of different lengths might be extracted. The approach adopted in this thesis is the following: if the input waveform exceeds a certain length t (in seconds), then a random subset length l_i is chosen ($i \in \{1, \dots, k\}$) and random start and end indices b and e are selected so that $e - b = l_i$, with $e \in \{0, \dots, s - l_i\}$ and $e \in \{l_i, \dots, s\}$. For example, if the waveform length exceeds $3\,\mathrm{s}$, then a random chunk of random length (selected in $[1, 1.5, 2]$ s) is extracted.

The final step in waveform pre-processing is that of **reverberation**, which is kind of the inverse process w.r.t. denoising, i.e. the goal is to make the audio sound as if it was uttered in a different environment. This step can be very useful for speaker/dialect verification tasks, to make models robust to noise and recognize the speaker/dialect even in wild scenarios, unseen during training. For the purpose of reverberation, this thesis adopts the convolution reverb approach, whereby the augmented signal is the result of a convolution between the clean signal and noise. Noise signals are extracted from a

database[9] of simulated and real Room Impulse-Response (RIR)[10], isotropic and point-source noises [90]. Figure 4.13 shows an example of a simulated RIR extracted in a small room, along with its processed version, where the main impulse is selected, the signal power is normalized and the time axis flipped; while Figure 4.14c shows the output of convolving the raw waveform in Figure 4.14a with the point-source noise in Figure 4.14b. In this example, the chosen noise resembles what can be heard when listening to house-builders using heavy machines.

Frequency-domain pre-processing

After having pre-processed data in the time domain, waveforms need to be converted to mel-scaled spectrograms, that can be further manipulated by frequency-domain transforms.

In order to perform the waveform to mel-spectrogram conversion, one needs to compute the STFT transform, which returns so-called **linear spectrograms** $spec_{lin}$, as a tuple of magnitude and phase. In order to get rid of phase, linear spectrograms are fully transitioned to the real domain as reported in Equation 4.2, where $spec_{lin}^{(r)}$ indicates the real part of the signal and $spec_{lin}^{(i)}$ its imaginary counterpart.

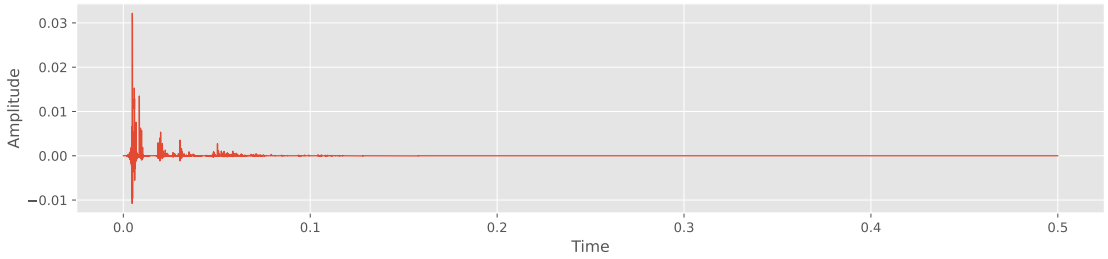
$$\sqrt{\left[spec_{lin}^{(r)}\right]^2 - \left[spec_{lin}^{(i)}\right]^2 + \epsilon}, \tag{4.2}$$

Next, linear spectrograms are converted to the **mel scale** to obtain $spec_{mel}$, amplitudes are mapped to the decibels range as $20 \cdot \log_{10}(spec_{mel})$ and the resulting output is possibly normalized as reported in Equation 4.3, where v_{max} represents the boundary value of the symmetrical range $[-v_{max}, v_{max}]$ where spectrogram magnitudes will be mapped to, and db_{min} is the minimum supported decibel value, i.e. db_{min} will be mapped to $-v_{max}$. The $\min$ and $\max$ functions serve the purpose of clipping outliers at the boundaries of $[-v_{max}, v_{max}]$.

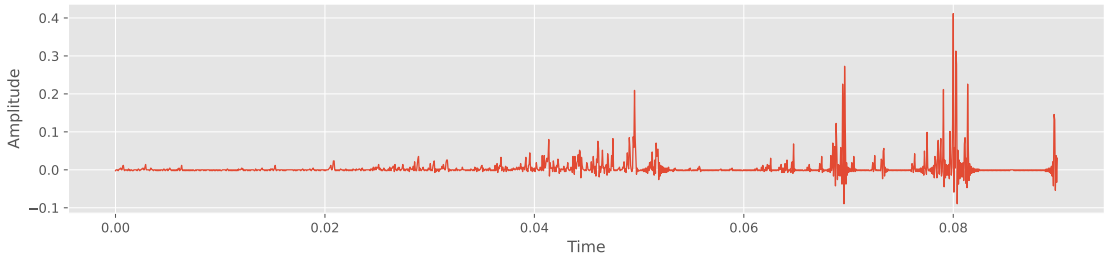
$$\min(\max(2 \cdot v_{max} \cdot \frac{spec_{mel} - db_{min}}{-db_{min}} - v_{max}, -v_{max}), v_{max}) \tag{4.3}$$

[9] https://www.openslr.org/28/

[10] An impulse response is defined as the sound of an acoustic space. It often starts as a recording of a short, sharp sound (the impulse) in the acoustic space in question, which excites the reverberation (the response) in the space.

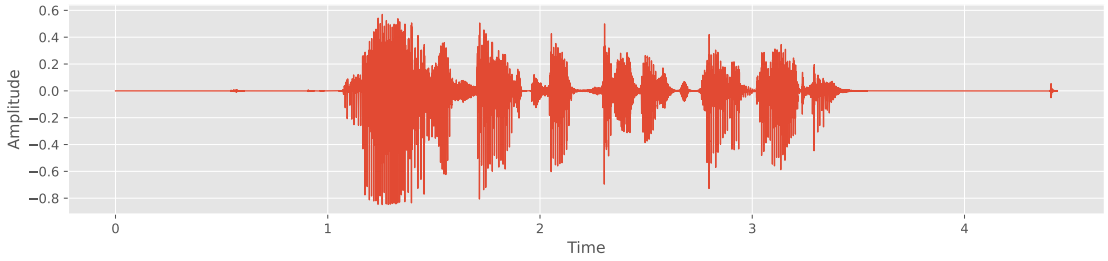


(a) Raw RIR, randomly extracted from the RIR database.

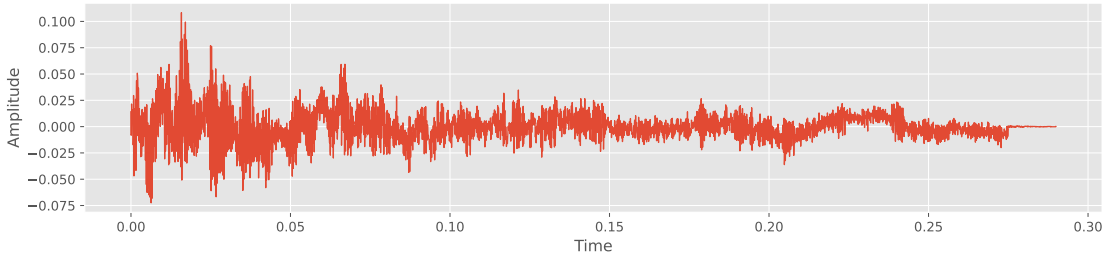


(b) Processed RIR, where the main impulse is selected and normalized and the time axis flipped.

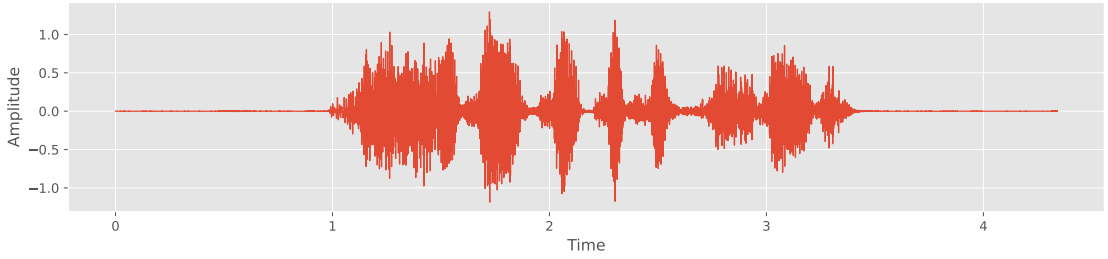
Figure 4.13: Example of a simulated RIR extracted in a small room.



(a) Raw waveform, to be corrupted by the noisy RIR.

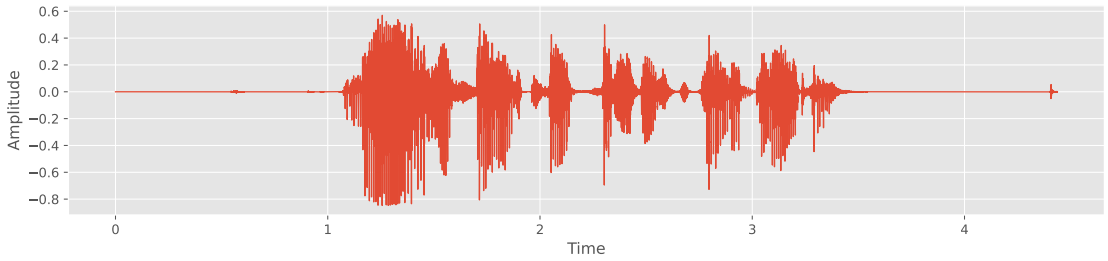


(b) Processed point-source noise, used to corrupt a clean audio signal.

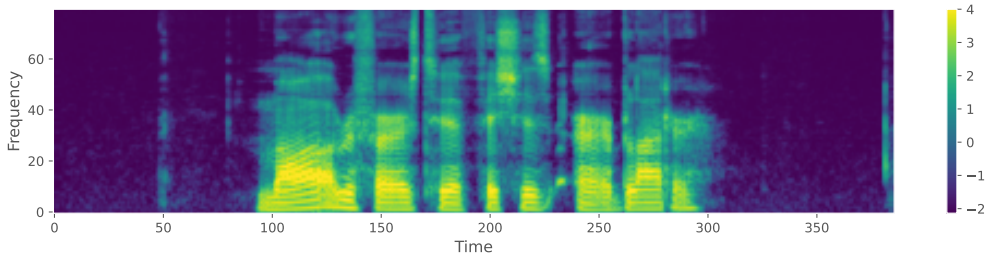


(c) Result of convolving the raw waveform with the selected point-source noise.

Figure 4.14: Convolution reverb, where a raw waveform (above) is convolved with a point-source noise (middle) to obtain a reverbed sound (below).



(a) Input waveform.



(b) Output spectrogram.

Figure 4.15: Waveform to mel-spectrogram conversion, through the STFT transform.

Figure 4.15 shows an example conversion from an input waveform to the corresponding normalized mel-spectrogram. Throughout the thesis, the waveform-to-spectrogram conversion will be based on parameters listed in Table 4.10.

After obtaining mel-spectrograms, other transforms may need to be applied. As in the time-domain, such further processing is usually introduced to perform strong regularization and allow DNN-based models to generalize well. One of the most popular transforms in the frequency domain is **SpecAugment** [135], a data augmentation method initially presented for the ASR field and later adapted for other tasks, such as speaker verification. SpecAugment is applied directly to mel-spectrograms and it consists of warping the features, masking blocks of frequency channels, and masking blocks of time steps. The intuition is that learned features should be robust to deformations in the time direction, partial loss of frequency information and partial loss of small segments of speech. These kinds of dropouts in the feature domain were already proposed in other fields such as Computer Vision, where Cutout [45] follows similar reasoning of randomly masking image patches. About SpecAugment, the time-warping transform can be seen as applying a speed perturbation to the input signal: the only difference w.r.t. to standard speed perturbation is that with SpecAugment it would be applied directly on the mel-scaled spectrogram, rather than on the raw waveform. The next two steps simply deal with

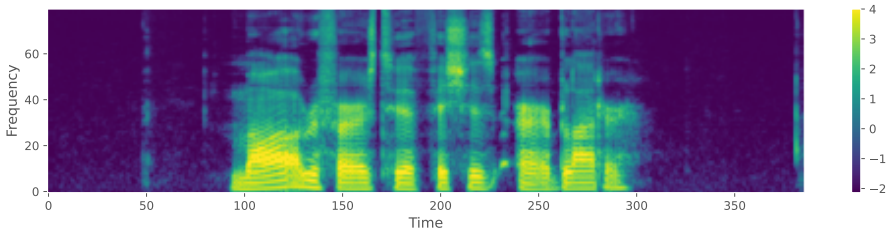
Parameter	Value	Description
n_f	1024	Size of FFT
n_s	$\lfloor \frac{n_f}{2} \rfloor + 1$	Number of bins in STFT
w	1024	Window size
h	256	Hop length
n_m	80	Number of mel bins
f_{min}	0.0	Minimum frequency
f_{max}	None	Maximum frequency
ϵ	10^{-5}	Clip linear spectrograms
mel_{scale}	Slaney	If Slaney, scale mels to have approximately constant energy per channel
mel_{norm}	Slaney	If Slaney, divide the triangular mel weights by the width of the mel band
p_v	$\lfloor \frac{n_f - h}{2} \rfloor$	Padding value
p_m	Reflect	Padding mode
$normalize$	True/False	Normalize in range $[-v_{max}, v_{max}]$
db_{min}	-100	Minimum decibel level
v_{max}	4.0	Maximum normalized absolute value
sr	22 050 Hz	Sampling rate

Table 4.10: Waveform to mel-spectrogram conversion fixed parameters.

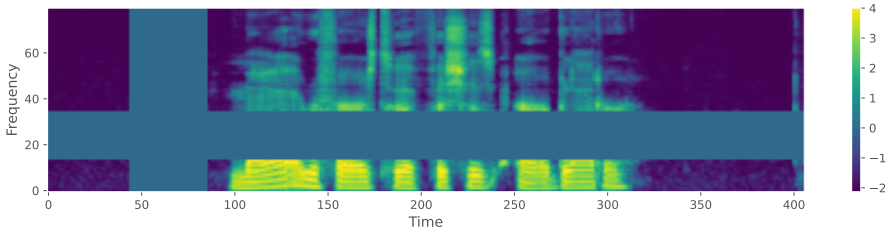
masking random consecutive patches of the spectrogram with a constant value. As described in the original paper, different patches may be applied to the time and frequency axes and multiple patches per axis may also be applied. Figure 4.16 shows an example of a SpecAugment transform with a random time warping in $[0.95, 1.05]$, 1 frequency mask with maximum coverage of 35% and 1 time mask with maximum coverage of 15%.

4.2.3 Splits

The last fundamental step in data pre-processing is that of splitting. As in all ML applications, data needs to be split into disjoint **training, testing and validation sets**. The training set is then used during the learning procedure and the testing set is used to assess models' performance on an unseen subset of data. In an ideal scenario, the training set should cover most (if not all) of the *true* data distribution, so that at test time the model



(a) Input spectrogram.



(b) Output spectrogram.

Figure 4.16: SpecAugment transform, by which frames in the input spectrogram (above) are randomly masked on the time and frequency dimensions to obtain an augmented spectrogram (below).

does not experience a loss in performance due to train-test time shifts. Instead, in DL the validation set usually serves the purpose of tracking model performance throughout the training procedure, but it can also be used to tweak the training time hyper-parameter, using the so-called **early stopping** procedure, whereby training is stopped (with some patience) when the validation error does not improve enough. This is also backed by Geoff Hinton, one of the *fathers of modern AI*: *Early stopping (is) beautiful free lunch.*[11].

In terms of **data splitting**, this thesis relies on the following reasoning. First of all, a subset of speakers is selected (if the dataset is a multi-dialect one, then the subset of speakers is taken over each dialect). Taking a subset of speakers is encouraged for the performance of certain audio conversion modules, such as AutoVC, while for audio embedding and TTS modules using the full set of speakers is a better alternative. Then, the training set is taken as either a certain percentage of utterances (e.g. 95%) for each speaker or all the utterances remaining after removing a fixed number of validation/test samples per speaker. The former option is used for TTS modules, while the latter one is for audio embedding and audio conversion modules. In this way, both the training set and the validation/test ones cover all selected speakers (and dialects, if applicable) in the dataset.

[11]NIPS 2015 tutorial slides, slide 63, http://www.iro.umontreal.ca/~bengioy/talks/DL-Tutorial-NIPS2015.pdf

CHAPTER 5

EXPERIMENTS

This chapter describes experimental details about all training and evaluation procedures. First, Section 5.1 gives an overview of the experimental setup we adopted, while Section 5.2 provides an explanation of the common evaluation strategy carried out for all modules in our TTS pipelines.

Then, the following sections report tests and analyses about each of its corresponding modules introduced in Chapter 3, with more details on the specific experimental setup needed. In particular, Section 5.3 deals with the speaker and dialect accent embedding modules, Section 5.4 details experiments related to the acoustic model, while experiments for the voice/accent conversion models are outlined in Section 5.5. Moreover, Section 5.6 is dedicated to the joint evaluation of the acoustic modelling and voice/accent conversion approaches, along with results obtained with the end-to-end approach described in Section 3.5.

Finally, the last section Section 5.7 summarizes the obtained results and provides discussion points and ideas for future work.

5.1 Experimental setup

The accompanying code[1] for this thesis was written in **Python**, a high-level programming language most popular in the science domain. The following is a non-comprehensive list

[1] https://github.com/Wadaboa/tts-dialects

of the libraries that were used in this project, where we only focus on the most important ones. First, we rely on **Librosa** [119], an open-source package for music and audio analysis, to compute the audio statistics reported in Chapter 4. For vector operations on CPU, such as the calculation of cosine similarities for speaker and dialect accent embedding models, we rely on **NumPy** [30], the de facto standard for scientific computing in Python. The **Pandas** [121] library was mainly used for its DataFrame data structure, to deal with Comma-Separated Values (CSV) files such as dataset annotations and evaluation results. All plots reported in this thesis were produced using either the **Matplotlib** [73] or **seaborn** [187] packages, thanks to their seamless integration with NumPy and Pandas, respectively.

For the implementation and training of neural models we rely on **PyTorch** [136], an open source DL framework with automatic differentiation and eager execution, and organize our code with its **PyTorch Lightning** [52] wrapper. To enable fast audio transforms on GPU, we use the **torchaudio** [195] library to load raw waveforms into PyTorch tensors and make use of some of their highly-optimized implementations, such as mel-spectrogram extraction. We also exploit **scikit-learn** [137] for training simple ML models, such as SVMs, and for the computation of certain evaluation metrics, such as ROC and F1.

All of the training and validation processes were executed on Google Colaboratory [61], also known as Colab, which is a platform that gives the possibility to exploit some computational resources for free; or the GPU cluster made available by the University of Bologna. For **Colab**, we always selected GPU runtimes, thanks to their Pro Plus subscription, and most of the time we were assigned an NVIDIA Tesla P100 GPU, with 16 GB of RAM. Sometimes, we were lucky enough to work with an NVIDIA A100 with 40 GB of RAM. Instead, the **university cluster** is a 10-node High-Performance Computing (HPC) cluster, where each node is equipped with an Intel Xeon quad-core CPU with 44 GB of RAM and an NVIDIA GeForce RTX 2080 Ti graphics card. Cluster computational resources are accessible by submitting requests to a SLURM service [76], which is an open-source workload manager that maintains a queue of jobs and schedules them according to a first-come-first-served rationale.

Some training runs, such as the ones executed through Colab, and all analysis of results were carried out via **Jupyter** [89] notebooks, which are available in the linked GitHub repository.

Training and evaluation metrics, along with model checkpoints and results, are directly logged into an openly-accessible **Weights & Biases (W&B)** project[2] [11]. W&B is a free experiment tracking, dataset versioning, and model management tool.

5.2 Evaluation strategy

All evaluations in subsequent sections are performed on specific test sets that vary between tasks and will be detailed afterwards. After the test-set is built and predictions are carried out using the model about to be evaluated, we compute module-dependent metrics. In particular, for the speaker and dialect accent encoders, we report results in terms of EER and minimum DCF, as explained in section Section 3.1, along with a thorough analysis of the latent space. Instead, acoustic models and voice/accent conversion systems share the same set of metrics.

Intelligibility is evaluated with a pre-trained ASR model, available through the **HuggingFace transformers** library [191]. The exact model we rely on is `facebook/s2t-small-librispeech-asr` [183], which is an end-to-end S2S transformer model trained on the full LibriSpeech corpus, where it achieves a WER of 4.3 on the *test-clean* split and 9.0 on the *test-other* split. Results for intelligibility are reported in terms of both WER and CER, along with detailed statistics for each (e.g. number of substitutions and deletions), and computed using the **jiwer**[3] library, without text concatenation. We also performed some evaluations with the Silero Speech-To-Text (STT) models[4] [167], which claim to achieve a WER of 6.1 on the *test-clean* split of LibriSpeech and 15.7 on its *test-other* split. The reason to consider Silero STT is that authors report objective metrics on the SLR83 dataset, with a minimum WER of 9.1 for Midlands female speakers and a maximum one of 25.3 for Irish male voices. Though, informal tests show that the previously mentioned HuggingFace model copes with dialect accents a lot better than Silero STT models; hence, all intelligibility-related results will be reported based on the former. Something worth pointing out is that intelligibility error metrics could stretch beyond 100%: this is possible as WER and CER are not bounded between 0 and 1, since a prediction could potentially return an infinite list of words or characters, respectively.

[2] https://wandb.ai/wadaboa/tts-dialects
[3] https://github.com/jitsi/jiwer
[4] All tests with Silero STT models were performed on the *xlarge* version of their community edition products and reported metrics refer to the EN V5 section of their quality benchmarks.

Next, we assess **naturalness** using a pre-trained MOSNet [112] model, publicly available through the **speechmetrics**[5] library. Given that the MOS score produced by MOSNet is a relative one, we provide a random ground-truth utterance of the speaker being evaluated when computing such scores. We are also aware that, in the speech community, MOSNet is not regarded as the most accurate solution to estimate naturalness, but we still rely on it for budget reasons, as running crowd-sourced tests would be too costly. To give readers a scale of reference for MOS scores, notice that the MOS for $16\,\text{kHz}$ natural speech is around 4.5, while the MOS scores of the current SOTA speech synthesizers are between 4 and 4.5 and scores for non-parallel VC are in the 3.0 to 3.8 range [104, 142], depending on the type of conversion (e.g. cross-gender is usually harder than in-gender).

The last objective metric we report is **speaker similarity**. To do that, we compute the average cosine similarity between test utterances and the centroid of the speaker being evaluated, using yet another pre-trained speaker encoder, namely the Emphasized Channel Attention, Propagation and Aggregation (ECAPA) Time-Delay Neural Network (TDNN) model [44] present in the **SpeechBrain** toolkit [143]. We rely on a different model than Resemblyzer and TitaNet so as not to bias evaluations since the speaker encoder in SpeechBrain is pre-trained on VoxCeleb1 [127] and VoxCeleb2 [28] and has thus not seen any of the identities in the datasets we use. We also tested an *x-vector* [157] based model from the same SpeechBrain package, but found it to consistently classify every utterance as close to any selected speaker. The simple test we run was to measure the cosine similarity between two utterances of the same speaker and between one utterance of a certain speaker and pure noise, expecting the former to return a cosine similarity close to 1 and the latter a value close to 0. What we found is that the x-vector model considered the pure noise sample as very similar to any speaker we selected, while the ECAPA TDNN model followed the correct reasoning, even though its cosine similarity values tend to be very conservative.

For what concerns **accentedness**, we perform perceptual tests through the Mechanical Turk crowd-sourcing platform [36]. We design the survey so that each worker is presented with the job description reported in Figure 5.1 and each sample is rated on a 1 to 5 scale, where 1 means the accent in the provided sample doesn't sound native at all, while a mark of 5 refers to a completely native accent. Moreover, tasks could only

[5] https://github.com/aliutkus/speechmetrics

Mark, on a scale of 1 to 5, how much the given speech sample is close to the displayed British regional accent (e.g. Irish)

Figure 5.1: Description of the Mechanical Turk accentedness evaluation, which is what workers see before accepting tasks in a batch.

be carried out by workers based in Great Britain, so as to obtain meaningful evaluations for our dialects of the British Isles. Each worker is paid $0.01\,\$$ per assignment and we request 10 assignments for each sample. Even though 10 ratings are probably not enough to obtain statistically significant results, we still go for 10 to limit expenses and because we consider this number to be enough to get an intuition regarding systems ranking.

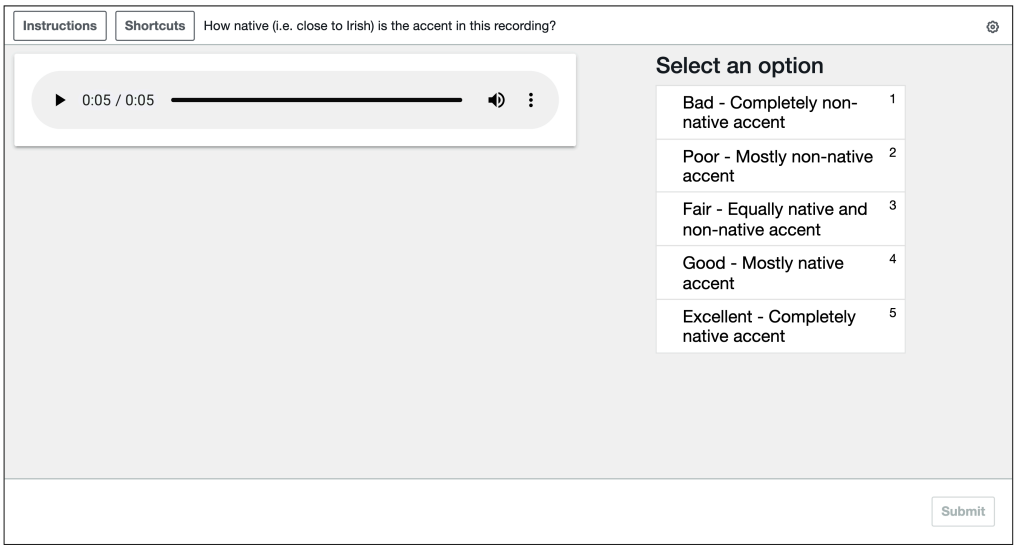


Figure 5.2: Example task from the Mechanical Turk accentedness evaluation. The worker is asked to evaluate how close the given sample sounds to the specified accent, on a scale of 1 to 5.

5.3 Speaker and dialect accent embeddings

Before diving deep into the speaker and dialect accent embeddings experiments, we need to clarify what's the **common ground** between each of them. In particular, the **data splitting** procedure is the one described in Section 4.2, i.e. whenever we need to split a dataset into training and validation utterances, we always consider all speakers/dialects in both splits and simply assign a fixed number of utterances for each speaker/dialect to the validation set: in this way, we are able to test generalization abilities of the models in terms of seen speakers/dialects, but unseen utterances. Further generalization in terms of unseen

speakers is verified for some models using completely different datasets. Also, all embeddings are reduced from their original dimension to 2D using Uniform Manifold Approximation and Projection (UMAP) [120], unless otherwise specified. For what concerns metrics, the minimum DCF is computed using $P_{target} = 0.01$ and $C_{FA} = C_{miss} = 1$, while both minimum DCF and EER adopt a cosine similarity backend.

In terms of **hyper-parameters**, all of our custom d-vector models follow specifications reported in Table 5.1, while all of our TitaNet models are trained according to parameters in Table 5.2. Both the d-vector and TitaNet models project input mel-spectrograms into embedding vectors of size 192. Moreover, both architectures are trained with the Adam [85] optimizer and a learning rate of 0.001 (no scheduler or weight decay were used). Both sets of parameters follow recommendations in the corresponding papers.

Parameter	**Value**
Number of LSTM layers	3
LSTM hidden size	768
Segment length	160
LSTM average	True

Table 5.1: D-vector hyper-parameters.

For the TitaNet model, some more details need to be discussed, as we deviate a bit from the original paper. In particular, using raw SGD makes the model diverge and we thus fall back to using Adam, as described above. Then, making use of all the **data augmentation** techniques described in the paper (i.e. chunking, SpecAugment, speed perturbation and reverberation) tends to regularize the model too much (at least in our small-scale experiments); hence, the only audio pre-processing step we apply is chunking, as introduced in Section 4.2. Also, the **number of mega blocks** for each model size and the rate of dropout were missing from the original paper. Regarding the former, 17 blocks were identified for TitaNet-S, while only 10 and 5 for TitaNet-M and TitaNet-L. Such numbers were computed by spawning 20 models for each size (S, M, and L), with a number of mega blocks between 1 and 20. Then, the final number of mega blocks was chosen by selecting models with the closest number of parameters to the ones reported in the paper, for each size (6.4 M for TitaNet-S, 13.4 M for TitaNet-M and 25.3 M for TitaNet-L). The only difference between model sizes is in the number of convolutional

filters used (256, 512 and 1024) and the convolutional kernel sizes (3, 5, 7). Finally, the last point to keep in mind is that our definition of mega blocks does not include the prolog and epilog blocks so the total number of blocks is actually the number of reported mega blocks plus 2.

Parameter	Value
Model size	S
Number of mega blocks	17
Attention hidden size	128
Dropout rate	0.1

Table 5.2: TitaNet hyper-parameters.

5.3.1 Speaker embeddings

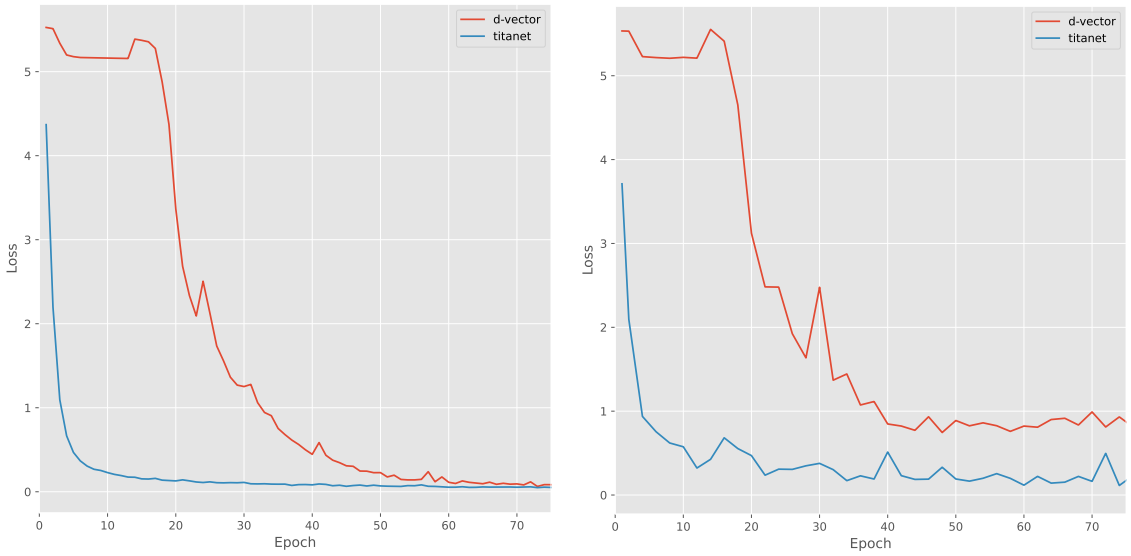
In this section, we test 3 speaker embedding models, namely our custom d-vector implementation, a pre-trained d-vector architecture and the TitaNet model, with the overall goal of finding the best speaker encoder to be used as part of the pipelines described in Chapter 3. The **high-level conclusion** is that in order to obtain effective embeddings, i.e. vectors that encode semantic information about the speaker, such as its gender, tone, average speaking rate and more, using large-scale corpora is advised, but in case such a massive dataset is not available (e.g. in low-resource languages) or there are not enough computational resources to train such models on so many utterances, exploring more sophisticated architectures might help.

D-vector vs TitaNet

First of all, we test our custom implementations of the d-vector and TitaNet models, by training both on the *train-clean-100* subset of **LibriSpeech** (see Subsection 4.1.2 for more details) and testing on LibriSpeech itself and the VCTK dataset (introduced in Subsection 4.1.3), in order to assess their generalization abilities. Both models were trained for the same number of epochs (75 in this example), using CE as loss function.

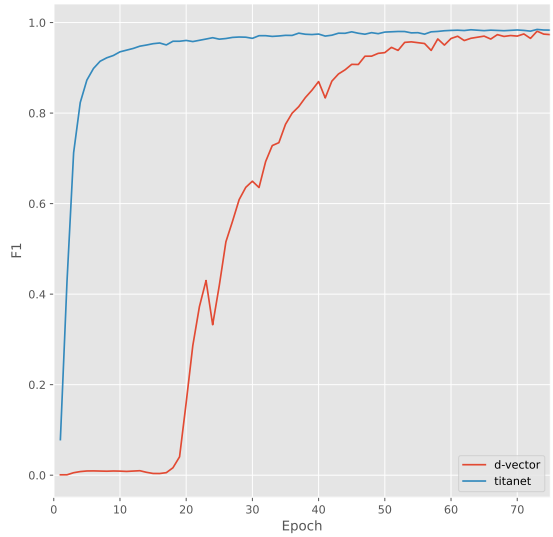
Figure 5.3 shows that training performance is comparable for both models, as they reach similar F1 (macro) values (and the same goes for accuracy), while validation metrics suggest that TitaNet outperforms the d-vector model on unseen utterances, by a good

margin. Moreover, Figure 5.3a and Figure 5.3b reveal that the d-vector model went into a slight **overfitting** path, given that the validation loss curve plateaus above the training one after about 40 epochs.

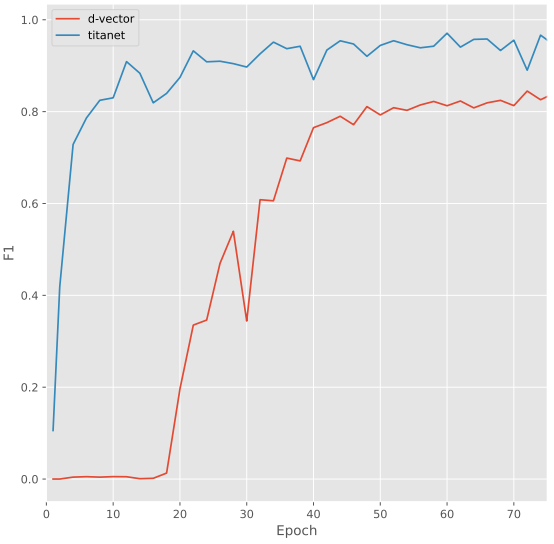


(a) Training loss.

(b) Validation loss.



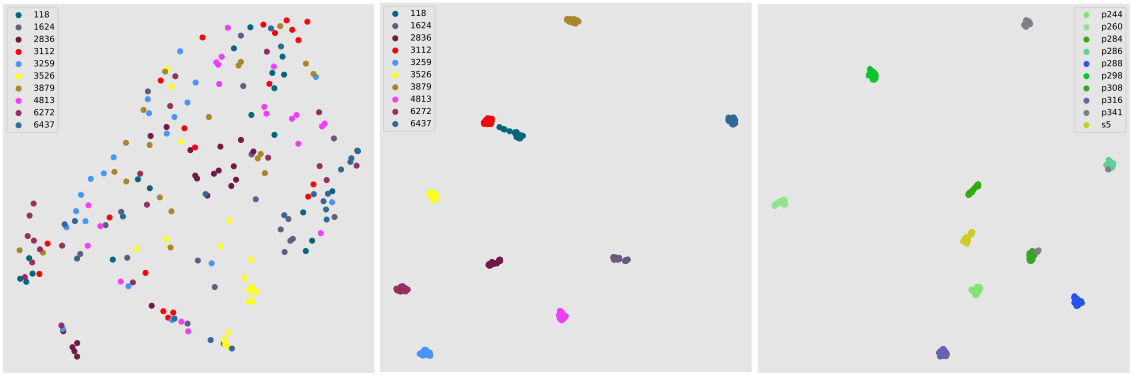
(c) Training F1.



(d) Validation F1.

Figure 5.3: D-vector (red) vs TitaNet (blue) loss (above) and F1 (below) curves on LibriSpeech. Results show that TitaNet outperforms the d-vector model on the unseen validation set.

Repeated empirical tests show that our d-vector model is not even able to correctly **cluster** the training data, as shown in Figure 5.4a. This could be due to the fact that the CE loss was used for training, which does not encourage the model to form visually (and, hopefully, semantically) meaningful clusters in the latent space, but only to discriminate between utterances of different speakers.



(a) D-vector tested on LibriSpeech.

(b) TitaNet tested on LibriSpeech.

(c) TitaNet tested on VCTK.

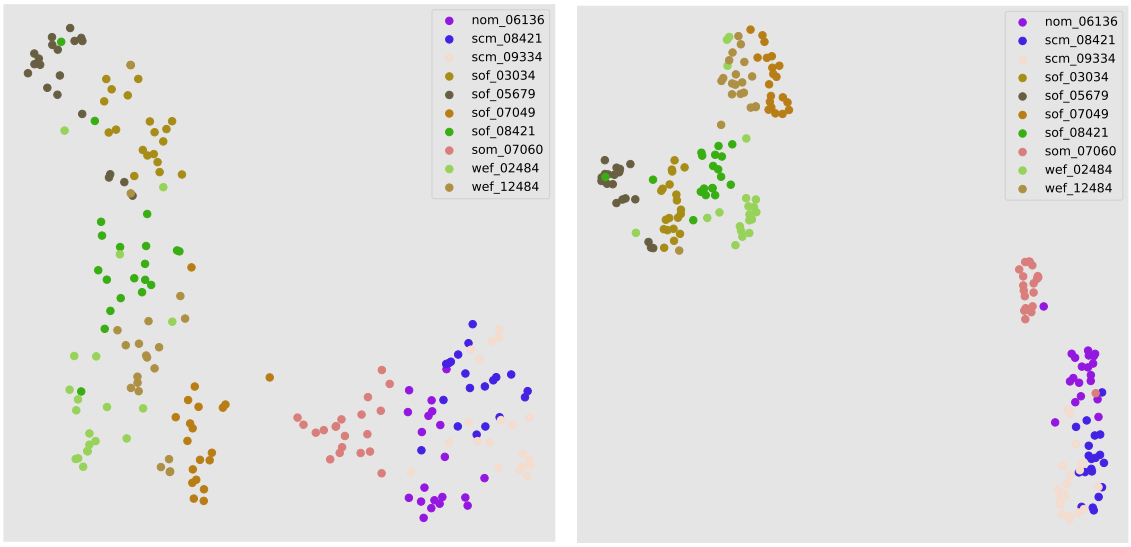
Figure 5.4: D-vector vs TitaNet speaker embeddings trained on LibriSpeech. Analyses of the latent spaces show that TitaNet outperforms the d-vector model on unseen utterances of both seen and unseen speakers.

Because of the poor results obtained in this analysis, we discard further tests on our custom d-vector model and focus on TitaNet, which instead correctly learned to form compact clusters (as shown in Figure 5.4b) of utterances, thus reflecting the high performance metrics obtained during training.

Moreover, we test **TitaNet on VCTK** by selecting a random set of 200 utterances (20 for each of 10 random speakers) and show clustering properties in Figure 5.4c. Next, we train a linear SVM to classify a random subset of 400 VCTK utterances (4 utterances for each of the 100 selected speakers) and we test the classifier on another random subset of 200 VCTK utterances (2 for each of the 100 speakers used in the fitting stage), to obtain an overall accuracy of 93%. We also compute the EER and minimum DCF metrics on the same set of VCTK utterances, giving us 9.54 and 0.80, respectively. As a reference, in the original paper, TitaNet authors are able to achieve an EER of 1.15 and a minimum DCF of 0.13 when testing TitaNet-S on VoxCeleb1 [127]: we attribute this gap in results to the huge difference in the number of utterances used to compute the metrics. Beware that these tests were carried out using the TitaNet checkpoint at 100 epochs, instead of the one at 75 epochs mentioned above. As we can observe, the model is able to generalize even to unseen speakers, at least for the speaker identification task.

To further test TitaNet capabilities, we repeat the same tests performed on VCTK for the **SLR83 dataset** (introduced in Subsection 4.1.4), given that it's the corpus we rely on for the core part of the thesis, i.e. multi-dialect experiments. Figure 5.5a shows results obtained when training TitaNet on LibriSpeech and testing it on SLR83, while Figure 5.5b plots the same speakers and utterances using a TitaNet model trained and

tested on SLR83. As we can see, both models obtain poor clustering properties, thus making us reconsider the use of TitaNet as our main choice for the speaker encoder module.



(a) Trained on LibriSpeech.

(b) Trained on SLR83.

Figure 5.5: TitaNet speaker embeddings tested on SLR83. Results show that, as expected, the model trained on the SLR83 dataset achieves the best clustering properties on the same corpus.

The conclusion of this experiment is that the main strengths of the d-vector model seem to come from the custom GE2E loss function and the size of the dataset used to train it, rather than the simplicity of its architecture. Instead, we observe that the scale of TitaNet seems to be beneficial for the model to more quickly learn effective clusters in the latent space, even with a standard CE loss and a limited number of utterances, despite the fact that results are highly dependant on the dataset selected for testing purposes.

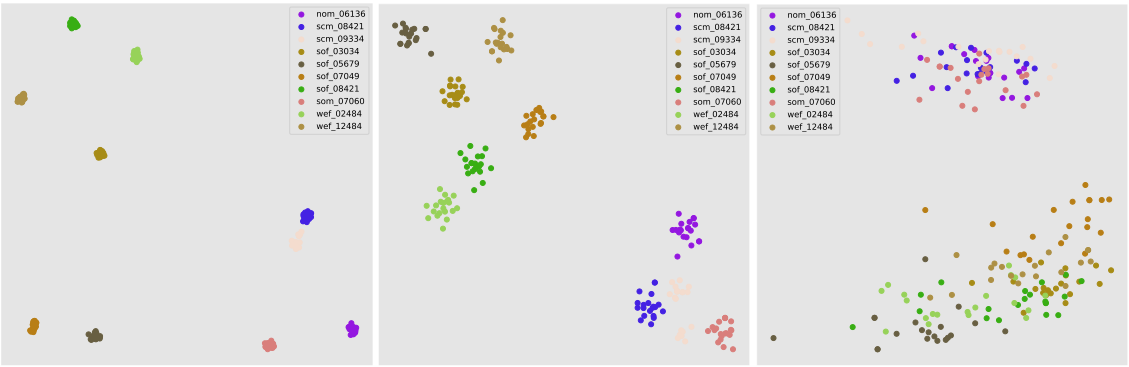
TitaNet vs Resemblyzer

This experiment acts as a follow-up to the conclusion of the previous experiment. The goal is now to compare TitaNet with a d-vector model trained with GE2E loss on a bigger dataset, to understand which one performs best on the SLR83 corpus. For this experiment we rely on a pre-trained d-vector-based encoder, publicly available under the name of **Resemblyzer**[6], which is trained with GE2E loss on the massive VoxCeleb2 [28] dataset[7].

[6] https://github.com/resemble-ai/Resemblyzer

[7] VoxCeleb2 contains over 1 million utterances for 6112 celebrities, extracted from videos uploaded to YouTube

Figure 5.6a shows clustering results obtained with the Resemblyzer model on SLR83. Embeddings plotted in Figure 5.6 come from the same set of 200 utterances (20 for each of the 10 selected speakers) used in Figure 5.5, so that a meaningful comparison with TitaNet could be visualized. Furthermore, we compute the EER and minimum DCF using Resemblyzer embeddings on the same subset of VCTK utterances used in the previous section to test TitaNet, obtaining 6.70 and 0.60, respectively: this leads us to believe that our TitaNet model actually performs quite well, even if it's trained on a fraction of the data used in Resemblyzer. Beware that the two reported EER and minimum DCF metrics are comparable, given that neither of the two models is trained with VCTK and the same set of test utterances is used to obtain such metrics.



(a) UMAP projection. (b) t-SNE projection. (c) SVD projection.

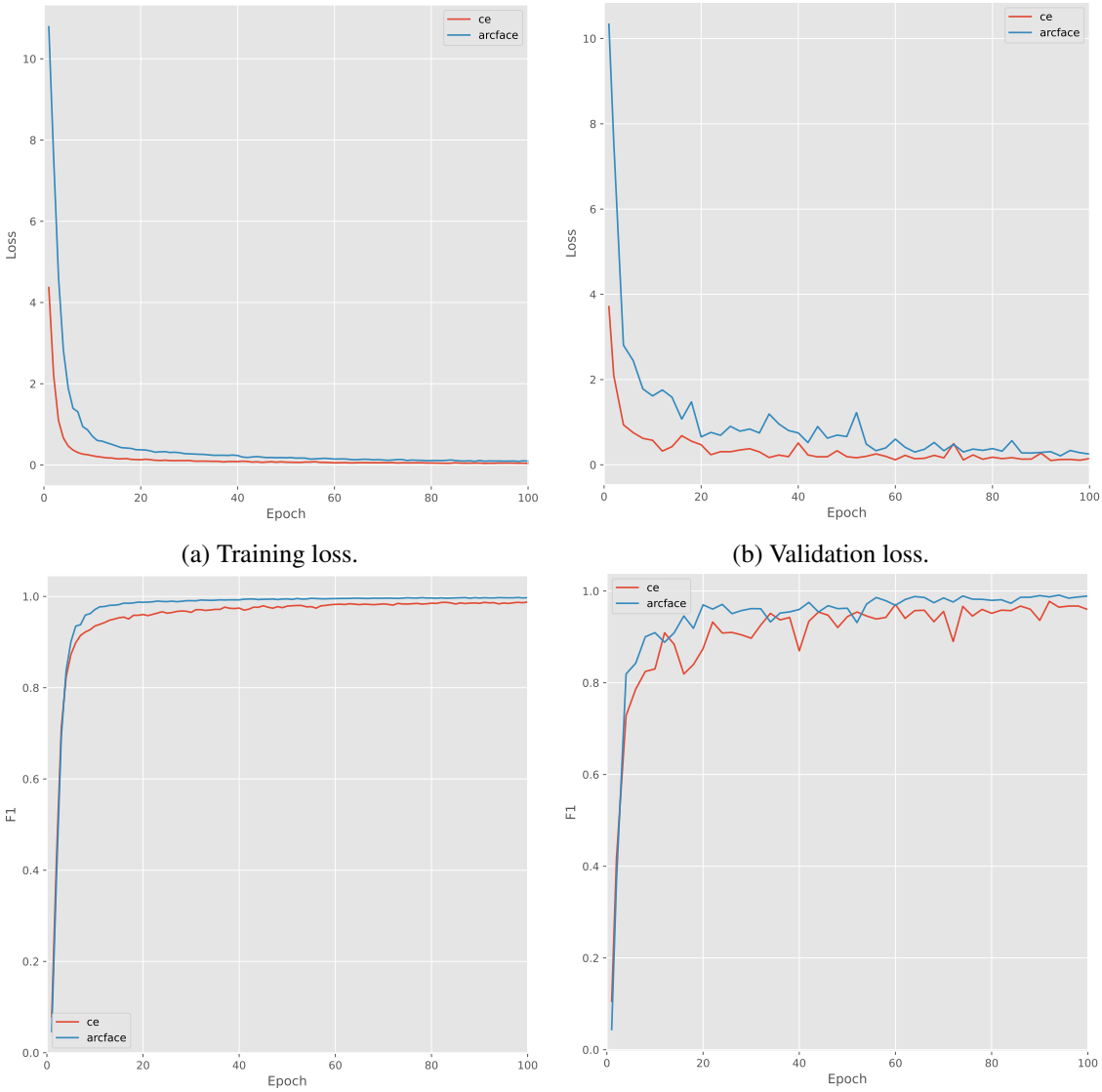
Figure 5.6: Resemblyzer speaker embeddings tested on SLR83. Results show that the choice of the 2D reduction algorithm has a noticeable effect on perceptual evaluations.

Figure 5.6 also shows the same embeddings plotted using different **2D reduction methods**: Figure 5.6a relies on UMAP, as all other embedding plots in this thesis, while Figure 5.6b makes use of t-distributed Stochastic Neighbor Embedding (t-SNE)[8] and Figure 5.6c applies the more conventional Singular Value Decomposition (SVD) reduction technique. As we can observe, the choice of the reduction method highly influences our perception in terms of the *goodness* of the embeddings. Following the literature, we settle for UMAP, as it seems to be the method that extracts the highest amount of information from our high-dimensional embedding spaces.

[8] t-SNE [117] is a tool to visualize high-dimensional data. It converts similarities between data points to joint probabilities and tries to minimize the Kullback-Leibler divergence between the joint probabilities of the low-dimensional embedding and the high-dimensional data. t-SNE has a cost function that is not convex, i.e. with different initializations we can get different results [137].

Loss function comparison

This experiment aims to understand the perceptual and objective differences obtained when training a speaker identification/verification model using **distinct loss functions**. In particular, we report results comparing a TitaNet model trained with standard CE loss and the same model trained with the ArcFace loss (also referred to as additive angular margin).



(a) Training loss.

(b) Validation loss.

(c) Training F1.

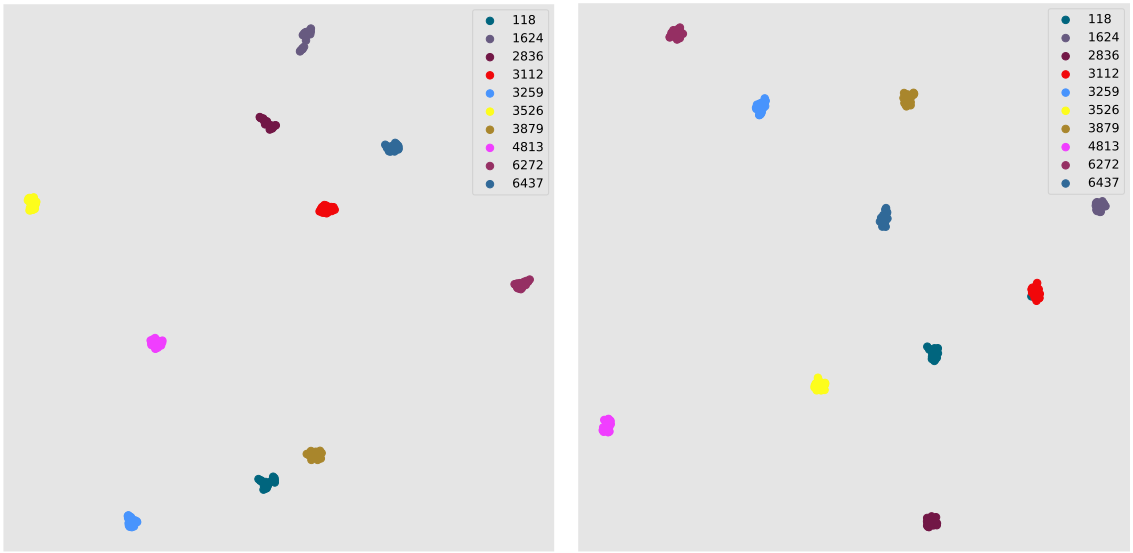
(d) Validation F1.

Figure 5.7: TitaNet CE (red) vs TitaNet ArcFace (blue) loss (above) and F1 (below) curves on LibriSpeech. Results show no noticeable different between the two models.

Given that TitaNet authors originally relied on **ArcFace**, we adopt the same set of hyper-parameters in our experiments, i.e. a margin $m = 0.2$ and a scale $s = 30$. It's worth pointing out that in the original TitaNet paper, authors report using $m = 30$ and

$s = 0.2$, thus relying on opposite magnitudes for margin and scale w.r.t. the original ArcFace parameters of $m = 0.5$ and $= 64$. Our experiments show that using $m = 30$ and $s = 0.2$ makes the model easily diverge; thus, we swap the margin and scale values following [43].

Figure 5.7 report training and validation losses and F1 scores for two TitaNet models sharing the same architectural design and dataset (LibriSpeech in this example), differing only in the selected loss function. The plots show **no noticeable differences** neither in terms of convergence speed, nor w.r.t. the reached performance levels. The two models also produce perceptually similar results, as shown in Figure 5.8.

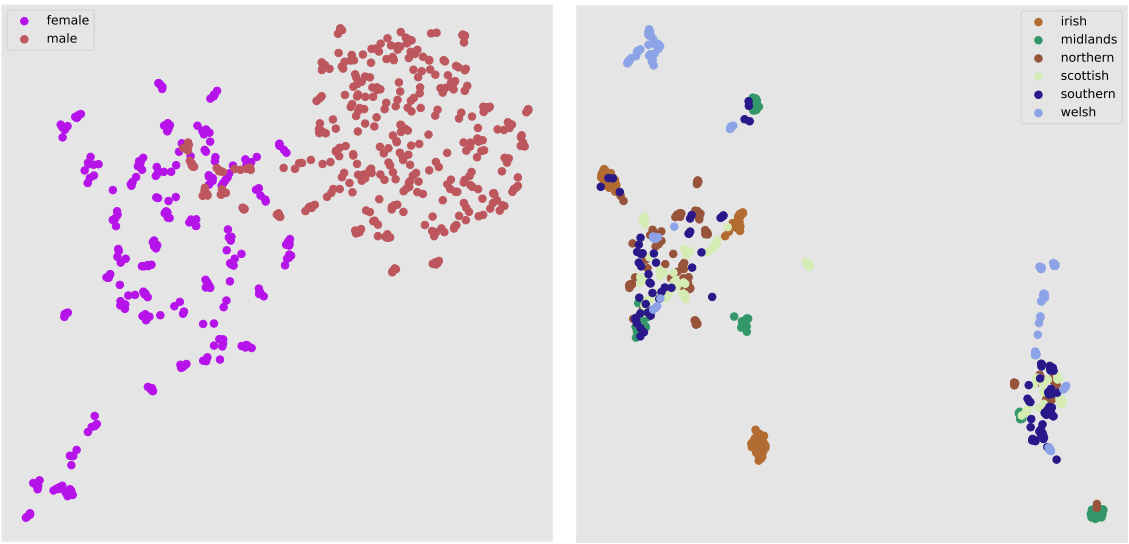


(a) CE loss. (b) ArcFace loss.

Figure 5.8: TitaNet CE vs TitaNet ArcFace speaker embeddings on LibriSpeech. Results show no noticeable different between the two models.

This experiment concludes that, at least in our use case, using TitaNet trained with CE loss or ArcFace loss makes no difference, as our goal is not strictly to generalize to unseen speakers. In such cases, we hypothesize that using ArcFace would be very beneficial, especially for datasets containing many more speakers than the ones we relied on. This is because the softmax operator used in CE would face a much harder discrimination task. Our hypothesis is also backed by applications in other domains, such as face recognition, where the use of metric learning losses starts to shine as the number of faces to recognize grows.

Extraction of high-level features



(a) Gender discrimination.

(b) Dialect discrimination.

Figure 5.9: Extraction of high-level features using Resemblyzer on SLR83. Results show that Resemblyzer embeddings encode gender information, but not regional accent.

In this experiment we aim to understand what kind of **high-level features** are encoded in our speaker embeddings. The first semantic feature we look for is gender, i.e. is the Resemblyzer model able to distinguish between **male and female** speakers in the SLR83 dataset? To answer this question, we randomly sample 600 utterances from SLR83, taking 5 utterances for each male speaker (71 male speakers in total, thus 355 utterances for the male group) and 5 utterances for each female speaker (49 female speakers in total, thus 245 utterances for the female group). Then, we compute Resemblyzer embeddings for each utterance, reduce them to 2D and assign colours based on gender. Figure 5.9a shows that the model is indeed able to identify speaker gender.

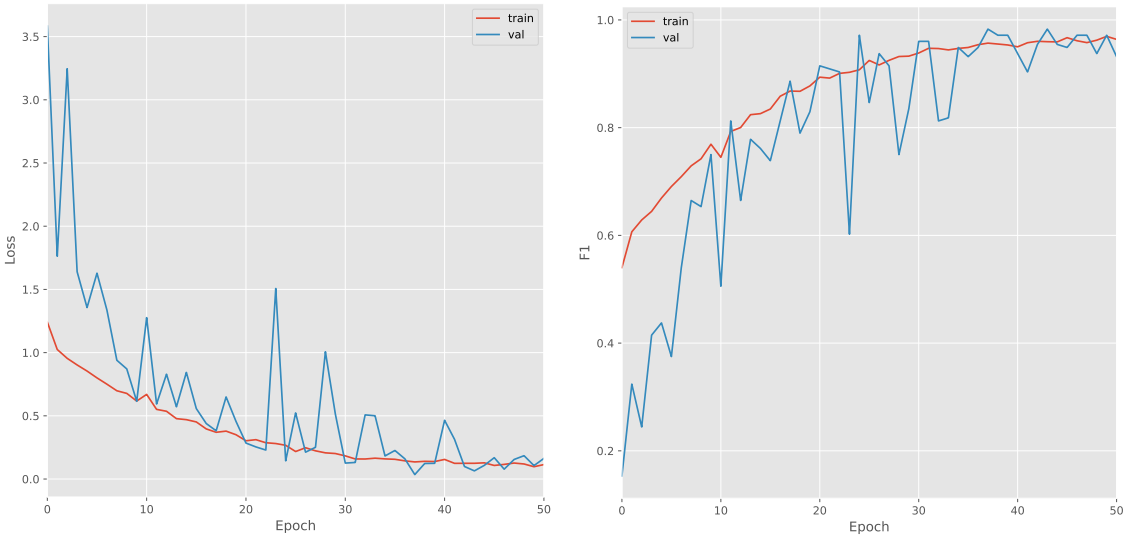
The next analysis adopts the same reasoning used above for gender, but with the objective of identifying **dialect accent**. To do so, we randomly sample 600 SLR83 utterances, 100 for each dialect (without balancing the number of utterances for speakers in the same dialect) and assign colours on the plot based on dialect. Figure 5.9b shows that Resemblyzer embeddings are not able to discriminate the regional accents present in the SLR83 dataset. We attribute this result to the fact that Resemblyzer was not specifically trained to account for regional accents, but we expect it to correctly cluster wider accents, such as British vs American ones. This result encourages us to use Resemblyzer as our speaker embedding model for downstream tasks, as it already seems to disentangle the

speaker and dialect accent information, which is a must-have property for the TTS and voice/accent conversion models described in Chapter 3.

5.3.2 Dialect accent embeddings

In this section, we present audio embedding models trained on the **dialect accent discrimination** task. Given the poor results obtained with our custom d-vector model for speaker embeddings and the fact that, to the best of our knowledge, no open-source work exists on the specific task of regional accent classification, we focus on adapting the TitaNet model and propose to use it as the dialect accent encoder for the pipelines described in Chapter 3.

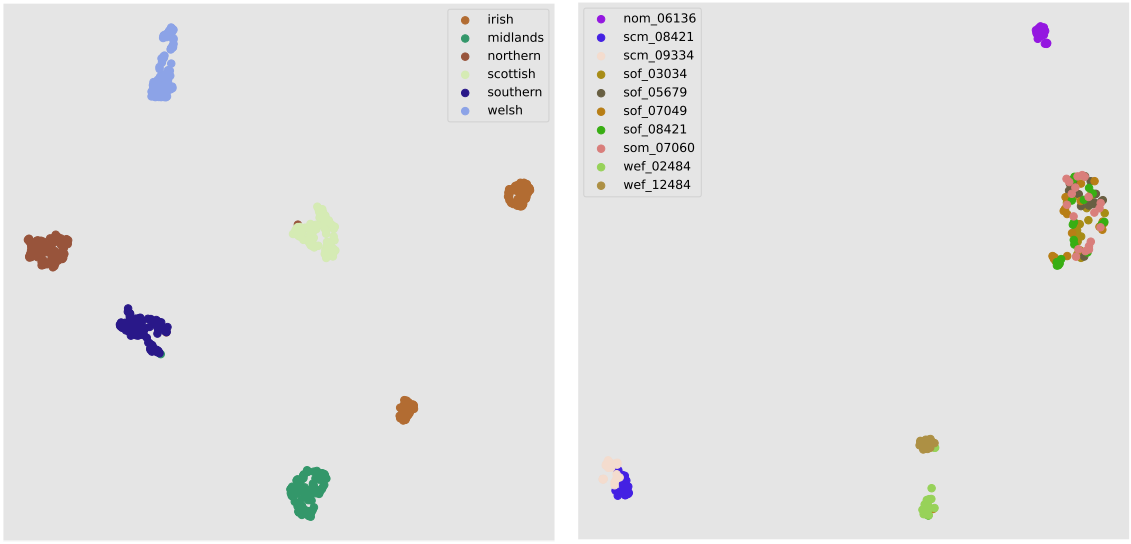
First, we train a TitaNet model on the SLR83 dataset, with the same architecture and hyper-parameters as the speaker encoders tested so far, by simply modifying its target labels from speaker to dialect indices. As shown in Figure 5.10, the model achieves more than 95% F1 on the validation set as early as 50 epochs, thus converging faster than its speaker counterpart. Moreover, Figure 5.11a shows the clustering properties of the model for all dialects in the SLR83 dataset. To give a meaningful comparison between our TitaNet dialect encoder and the Resemblyzer speaker encoder, Figure 5.11a uses the same utterances plotted in Figure 5.9b.



(a) Training and validation losses.

(b) Training and validation F1 scores.

Figure 5.10: TitaNet CE dialect embeddings metrics on SLR83. Results show good generalization properties as soon as 50 epochs, even though validation metrics exhibit more variability than the ones reported for speaker identification.



(a) Dialect discrimination.

(b) Speaker discrimination.

Figure 5.11: Speaker and dialect discrimination with TitaNet dialect embeddings on SLR83. Results show that dialect accent embeddings contain some speaker information, given the high correlation between speaker identity and dialect accent.

Second, we replicate some of the experiments carried out for speaker embeddings and reach the same conclusions. In particular, using loss functions different from standard CE does not seem to help.

Finally, to understand whether or not our dialect accent embeddings are **disentangled** from speaker information, we consider the same utterances used in Figure 5.5 and Figure 5.6 but compute embeddings using our trained dialect accent encoder. As shown in Figure 5.11b, speakers from the same dialect are nicely clustered together. Moreover, for some dialects (such as the Southern one on the top right), utterances from the same speaker do not seem to form mini-clusters within the dialect mega-cluster. Instead, for other dialects (such as the Welsh one on the bottom right) we observe the opposite trend, whereby speakers from the same dialect form sort of intra-clusters.

This latent space analysis leads us to believe that some speaker information is part of the dialect accent embeddings. We hypothesise that this mix-up happens due to the **high correlation** between speakers and dialects, i.e. each speaker only speaks one dialect. Multiple approaches may be tested to further disentangle the two types of features, such as **adversarial training** or **multi-task learning** [17]. For the former, an additional loss term is added to the standard classification one, to discourage the classifier to encode speaker or dialect accent information, for the dialect and speaker encoders, respectively. For example, if the objective is to train a dialect accent classifier, the adversarial loss

term should make sure the classifier is not able to discriminate speakers using the dialect accent latent representation. In this thesis, no adversarial training strategies have been employed, but some tests have been carried out in a multi-task learning scenario. Inspired by previous work in the multi-lingual setting [54], in our experiments a single TitaNet model was trained in a **siamese** fashion to discriminate both speakers and dialects. In particular, the model was configured to share most of the building blocks, up to the final bottleneck and classification layers. At that point, the model splits in 2 heads, one for the speaker and one for the dialect accent information. In terms of losses, the multi-task objective is the sum of speaker and dialect accent CEs. In this way, the model is encouraged to share common representations up to the splitting point, which then encodes the inductive bias of making the two types of embeddings deviate from each other. Results reveal that no noticeable improvements regarding objective metrics and disentanglement properties can be observed. Thus, we discard the usage of this multi-task model, but still report it as it achieves very similar results when compared to the single speaker and dialect accent encoders while relying on almost half the number of parameters, which might be an important attribute to consider in resource-constrained scenarios.

5.4 Acoustic model

For what regards experiments on acoustic modelling (i.e. converting text to an intermediate speech representation), we first test our custom Tacotron 2 implementation on a single speaker dataset (LJ Speech) and report results in Subsection 5.4.1. Then, in Subsection 5.4.2 we extend the architecture to work with multiple speakers and multiple dialects.

All Tacotron 2 trainings share the **hyper-parameters** reported in Table 5.3, following [152]. Other hyper-parameters that were changed between different training runs are the batch size (usually set to 16 because of memory constraints), the use of speaker/dialect embeddings and their dimensions (usually 256 for speaker embeddings and 192 for dialect embeddings), the use of learning rate schedules (along with their associated configuration) and the reduction factor (alternated between $r = 2$ and $r = 5$). The learning procedures were carried out using the Adam optimizer [85] with $\epsilon = 0.000001$. Moreover, we rely on masked loss functions (for both the reconstruction and the stop token

criteria) to avoid back-propagating through the padding values that were added to construct mini-batches.

5.4.1 Single-speaker

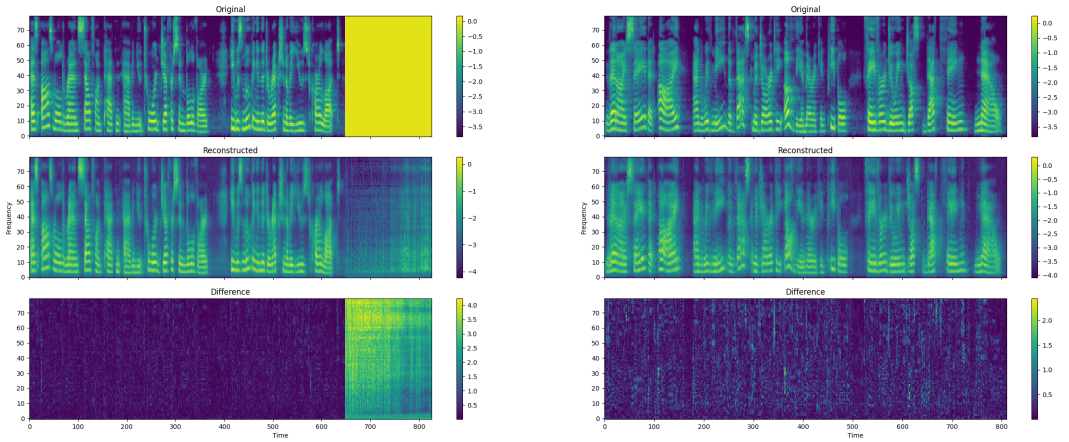
The main objective of training a single-speaker model is to assess the correctness of our custom implementation and understand the role played by different hyper-parameters in terms of convergence speed and output quality. Note that all single-speaker trainings were carried out on the **LJ Speech** dataset, introduced in Subsection 4.1.1.

One of the main findings from our experiments is that the **attention block** in Tacotron 2 brings major instabilities, both at training and at inference time. Regarding the former, it is easy for the model to follow a wrong optimization path at the very beginning of training and never recover: this is shown by the alignment between the encoder (i.e. the text inputs) and the decoder (i.e. the mel-spectrogram frames) not following a mostly diagonal (or, more generally, monotonic) path. In such an unfortunate case, the training run is unrecoverable and a new experiment needs to be started for further evaluation. The key factors for learning correct **alignments** seem to be the reduction factor and the silence trimming pre-processing step, whereby silence regions are removed at the beginning and end of raw waveforms, before transitioning to the frequency domain (as described in Section 4.2). For what concerns the **reduction factor**, which intuitively encodes the number of mel-spectrogram frames to output at each auto-regressive step, we found that relying on relatively high values of r (such as $r = 5$) makes the model learn attention alignment as soon as $8\,\mathrm{K}$ steps, but produces overly-smoothed spectrograms, while using lower reduction factors leads to a slower alignment with better overall quality. We report an example of such behaviour in Figure 5.12, where predictions in Figure 5.12a are produced from a model trained for around $40\,\mathrm{K}$ steps and $r = 2$, while predictions in Figure 5.12b are the output of a model trained for $75\,\mathrm{K}$ steps and $r = 5$. As can be observed, the model with the lower reduction ratio generates crispier mel-spectrograms in about half the number of training steps. Notice that both models were trained with the same hyper-parameters, using the Noam learning rate scheduler and both predictions were performed on the same unseen validation set, but on different utterances, and relying on the teacher forcing scheme.

As a further example, Figure 5.13 shows a mel-spectrogram produced by our Tacotron

Parameter	Value
Learning rate	0.001
Gradient norm clip value	1.0
Symbols embedding size	512
Encoder number of convolutional layers	3
Encoder convolutions kernel size	5
Encoder recurrent unit	LSTM
Encoder number of recurrent layers	1
Encoder recurrent bi-directional	True
Decoder state size	1024
Decoder recurrent unit	LSTM
Decoder number of recurrent layers	2
Attention hidden size	128
Attention location number of convolutional filters	32
Attention location convolutions kernel size	31
Attention sharpening	False
Attention sharpening beta	1
Attention smoothing	False
PreNet hidden sizes	[256]
PostNet number of convolutional layers	5
PostNet hidden size	512
PostNet convolutions kernel size	5
Dropout	0.45
Reconstruction loss weight	1.0
Stop prediction loss weight	1.0

Table 5.3: Tacotron 2 hyper-parameters.



(a) Using $r = 2$.

(b) Using $r = 5$.

Figure 5.12: Predictions for Tacotron 2 trained on LJ Speech with $r = 2$ and $r = 5$. The model with the lower reduction ratio tends to produce crispier spectrograms.

2 model trained on LJ Speech for around 200 epochs[9], a reduction factor of $r = 5$ and a constant learning rate of 0.001. As we can see, the mel-spectrogram is somewhat blurry (given the fact that we used $r = 5$), but informal listening reveals that the vocoded speech sounds natural, with minor quality issues, for the reported piece of text and other unseen utterances. Something worth pointing out is that our model was not able to learn effective **stop token** probabilities, meaning that the produced mel-spectrograms are longer than they should. This is also mirrored in Figure 5.13, where the entire sentence is uttered by frame 420, but our model keeps generating frames until the 650 frame mark.

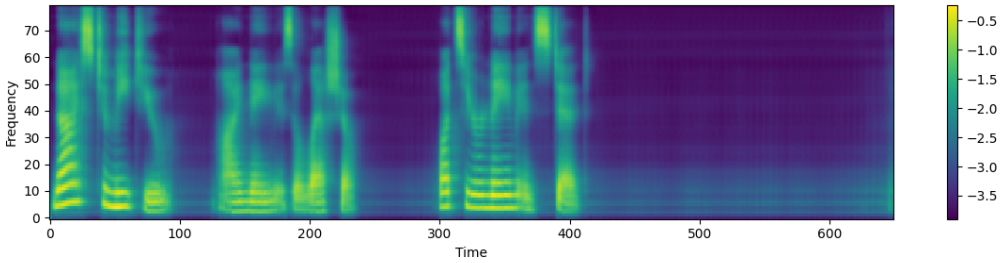


Figure 5.13: Tacotron 2 at inference time for input text: *Nice to meet you, my name is Alessio. What's your name instead?*.

Moving to **formal evaluations**, we extract 100 random prompts from the *"ax"* subset of the open-source GLUE dataset [182], which contains manually-curated entries for fine-grained analysis of system performance on a broad range of linguistic phenomena. This dataset is usually the benchmark for sentence understanding tasks in the NLP field,

[9] It took around 3 days of training on an NVIDIA RTX 2080 Ti.

but we only use it for its high quality and for the fact that it's publicly available. After having extracted such random utterances, we perform predictions and evaluate them on the speaker similarity, naturalness and intelligibility dimensions, as explained in Section 5.2. What we observe is that **intelligibility** (in terms of WER and CER) appears to be relatively low, even though the vast majority of samples sound very comprehensible through informal listening. Something we noticed is that the voice is not completely clean, meaning that some artefacts are produced by the neural synthesiser, and we hypothesise that such artefacts tend to harm the ASR models used to compute intelligibility metrics. To verify our assumption, we pass all samples through the **speech enhancement** model reported in Section 4.2 and observe non-negligible improvements in terms of clarity, which are not reflected in objective measures. Counter-intuitively, the denoised samples achieve much lower WER and CER values than the noisy ones, even though perceptual tests hint at the opposite trend. Thus, if we were to follow subjective evaluations, speech enhancement models could also be used as part of the TTS pipeline (after the vocoding stage) to enable an overall training time reduction. Results for both the noisy and denoised samples are reported in Table 5.4. For reference, for both evaluations, we compute the LJ Speech single-speaker centroid by averaging speaker embeddings over 100 ground-truth audio snippets.

	Speaker similarity (Cosine)	**Naturalness (MOS)**	**Intelligibility (WER)**	**Intelligibility (CER)**
Noisy	52.72 ± 8.03	2.56 ± 0.21	119.20	86.01
Denoised	46.79 ± 6.52	2.76 ± 0.12	144.70	108.15

Table 5.4: Single-speaker acoustic modelling results on LJ Speech.

Using the Noam scheduler

The second experiment we perform is related to the usage of a **learning rate scheduler**. The most popular scheduler for the task of acoustic modelling seems to be the **Noam scheduler**, which was introduced in [177] for the task of Machine Translation (MT). The Noam scheme is defined by a linear **warmup phase** for a given number of steps, followed by an **exponential decay** where the learning rate decreases proportionally to the inverse square root of the step number (scaled by the inverse square root of the dimensionality of the model), as shown in Figure 5.14 and reported in Equation 5.1. Note that d in

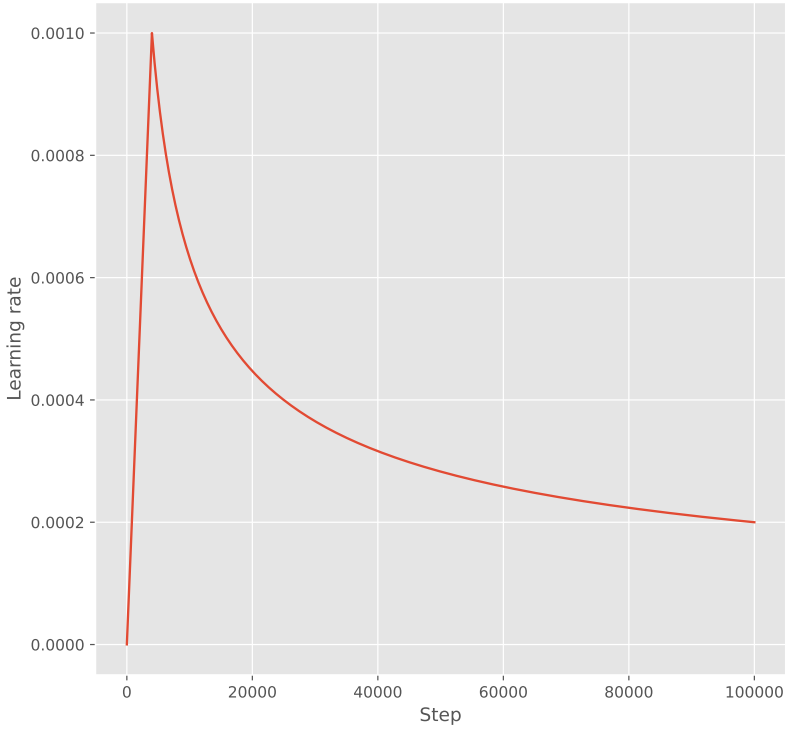


Figure 5.14: Noam decay over 100 K steps with a maximum learning rate of 0.001.

Equation 5.1 corresponds to the dimensionality of the model, which can be interpreted as the hidden size parameter which dominates the number of parameters of the model, while t is the current number of steps and w is the number of warmup steps.

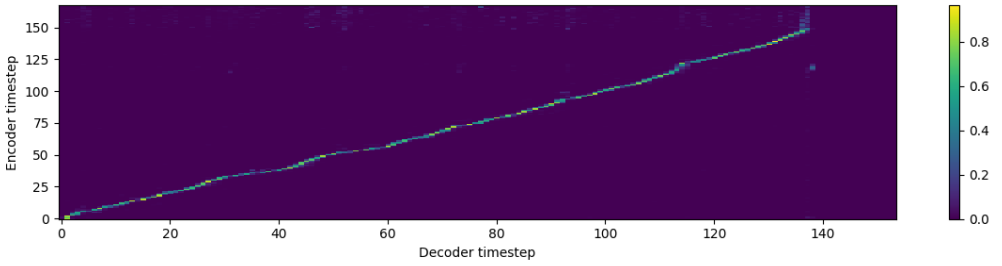
$$lr_{t+1} = d^{-0.5} \cdot \min\{t^{-0.5}, t \cdot w^{-1.5}\} \tag{5.1}$$

Our implementation slightly deviates from [177], as we rely on public code from Coqui TTS that uses Equation 5.2, where $d^{-0.5}$ is replaced by $lr_t \cdot w^{0.5}$. Following [177], for all experiments the number of warmup steps w is set to 4000.

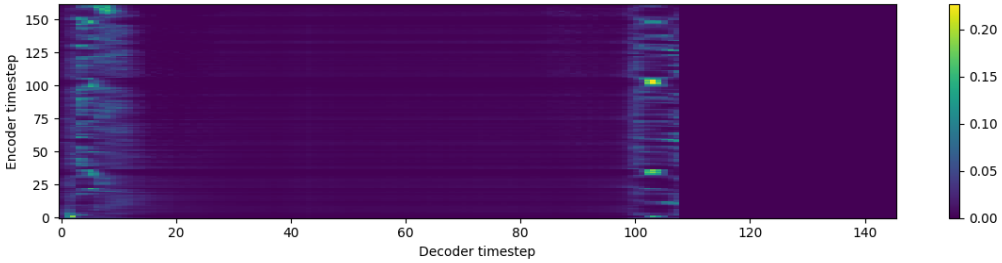
$$lr_{t+1} = lr_t \cdot w^{0.5} \cdot \min\{t^{-0.5}, t \cdot w^{-1.5}\} \tag{5.2}$$

Unfortunately, our results indicate much **slower convergence** when using the Noam scheduler, w.r.t. relying on a simple constant learning rate. Figure 5.15 shows attention alignments extracted from different random validation utterances. In particular, the attention map in Figure 5.15a is selected at around 8 K steps, while the one in Figure 5.15b at almost 70 K steps. Repeated experiments replicate the exact same behaviour, whereby Tacotron 2 with the Noam scheduler doesn't learn to focus its attention correctly, even

when trained for longer. Thus, we discard the usage of Noam for further experiments.



(a) Without Noam.



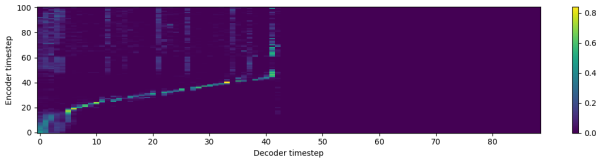
(b) With Noam.

Figure 5.15: Attention alignment for Tacotron 2 trained on LJ Speech with $r = 5$.

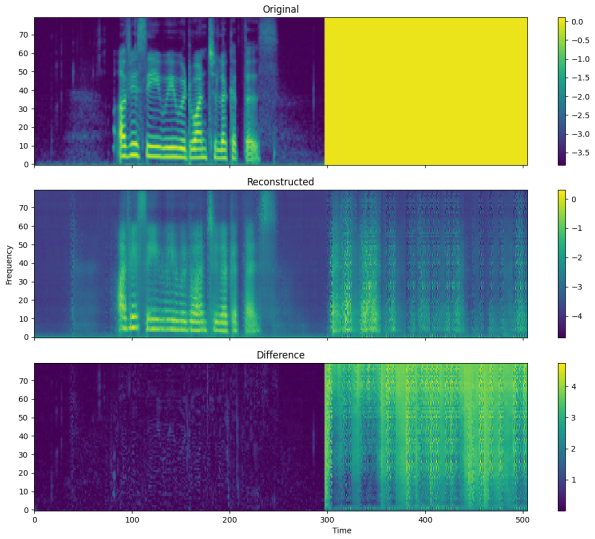
5.4.2 Multi-speaker and multi-dialect

In this section, we extend the single-speaker Tacotron 2 experiments to the **multi-speaker** scenario. In order to do so, we explore conditioning Tacotron's encoder outputs in 2 ways, i.e. either using pre-trained speaker embedding modules or by relying on learned lookup tables, so as to let the model learn speaker representations as part of the overall training procedure. All experiments in this section were performed with $r = 5$, a constant learning rate and un-normalized spectrograms.

First, we train our multi-speaker acoustic model on the **VCTK dataset** and obtain results shown in Figure 5.16. Note that this experiment relies on Resemblyzer's speaker embeddings and does not trim silence regions in raw waveforms as part of the data pre-processing steps. As can be observed, acceptable alignment and predictions are produced as early as $40\,\mathrm{K}$ steps. Informal listening shows that the model is able to produce intelligible speech, even though it falls short on naturalness. Nevertheless, we assume that training for longer would make the synthesis sound less robotic, thus also increasing the perceived similarity with the selected speaker.



(a) Attention alignment.



(b) Teacher-forced prediction.

Figure 5.16: Outputs from multi-speaker Tacotron 2 trained on VCTK.

Next, we train the same model on the **SLR83 dataset**. For this scenario, a multitude of experiments was carried out and most of them didn't work out as expected. For example, we were expecting **silence removal** to be vital for the model to pick up alignment, but what we found is that, specifically for the SLR83 dataset, training the acoustic model when starting from waveforms where small regions of silence are left as-is at the beginning and end of utterances improves convergence speed.

Other experiments that didn't bring quality improvements are, for example, setting different weights for different entries in Tacotron's loss function. Given that one of the main issues in attention-based acoustic modelling is the trade-off between convergence speed and quality, we hypothesise that relying on a high reduction ratio and a **high reconstruction loss weight** might result in the best of both worlds. In particular, we train a multi-speaker Tacotron 2 model on SLR83, with a reduction factor $r = 5$ and a reconstruction weight of 2 for both pre-residual and post-residual loss entries. Unfortunately, the intuition does not seem to empirically hold, as no improvement in the quality of predicted spectrograms could be observed from visual inspection.

Furthermore, we tested optimization strategies popular in the open-source speech

community, but none of them brought good enough improvements to be included in our standard training recipe. Some examples include using a teacher forcing scheduler and dropping ground truth spectrogram frames as a form of regularization.

Regarding **teacher forcing**, lots of work has been done to deal with the train-test mismatch auto-regressive models have to face. For example, Lamb et al. explore a GAN-based approach, whereby a discriminator is trained to distinguish between sequences generated using teacher forcing and scheduled sampling, while a generator is trained to fool the discriminator, nudging the dynamics of teacher forcing and scheduled sampling to become more similar. Our approach to a modified teacher forcing is simpler, yet effective. Indeed, we rely on a decay mechanism for which the model slowly transitions from a full teacher-forcing training procedure to a free-running one, thus relying more and more on its own predictions as training goes on. Intuitively, this scheduling procedure should be effortless, but in our experiments, we find it hard to select the best decaying mechanism. If the decay is too fast then the model relies too early on its noisy predictions, while if the decay is applied late in the learning phase no improvements are observed with such scheduling behaviour. Thus, setting the right decay is not straightforward and requires careful tuning. Hence, we exclude this technique from further experiments.

For what concerns **spectrogram frames dropping** [111], its main objective is to increase the information gap between the teacher forcing input and target, which has been shown to be vital for Tacotron to achieve acceptable performance. The way spectrogram frames dropping works is by discarding teacher forcing input frames randomly to a certain percentage, by setting them to the global (per-speaker) mean. This technique is supposed to boost up alignment learning a lot when the dropping rate is set to relatively low values such as 0.1 or 0.2. In terms of implementation, we follow the original open-source code[10]. Again, results are not satisfactory enough when testing this technique on the SLR83 dataset. Hence, as with teacher forcing scheduling, we also exclude spectrogram frames dropping from further experiments.

The last experiment worth mentioning is about comparing the multi-speaker acoustic model trained with frozen speaker embeddings and **one-hot encodings** of speaker indices. For the speaker embeddings, we rely on per-speaker vectors, extracted as the centroids of all utterances from a given speaker in the training split of SLR83, while for one-hots we simply rely on a to-be-trained per-speaker lookup table. What we observe is

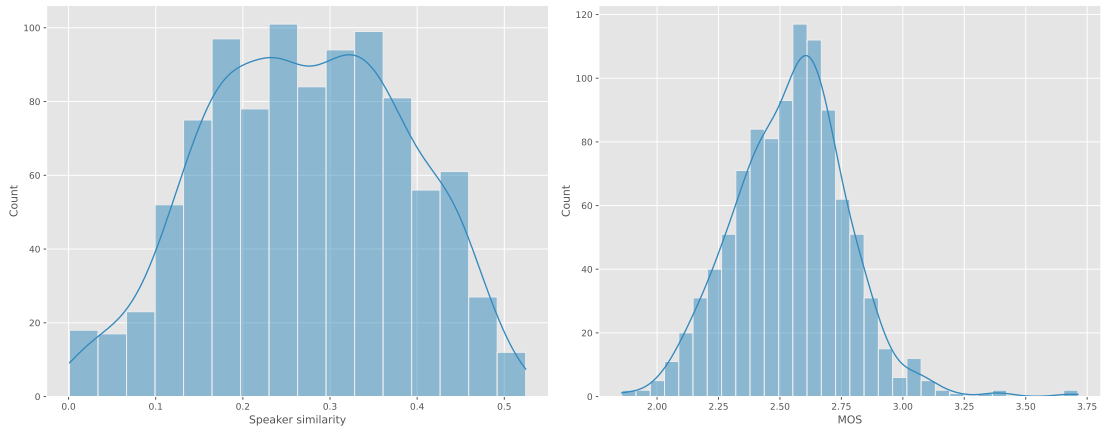
[10] https://github.com/bfs18/tacotron2

that using pre-trained speaker embeddings leads to a faster alignment learning process, as the model with one-hot encodings needs more training time to learn meaningful speaker representations, and while it learns such representations the predicted mel-spectrograms are more or less pure noise. We hypothesise that the one-hot model could easily become on par with the speaker embeddings one, in terms of convergence speed, if we were to rely on **auxiliary losses**, such as a simple CE to ensure that learned representations encode enough speaker information to be able to perform good enough classification. In the interest of time, we leave such experiments for future work.

Finally, to reduce experimentation time and obtain faster convergence, we test a **fine-tuning** strategy, whereby Tacotron 2 is pre-trained on the LJ Speech mono-speaker dataset for a certain number of training steps, and after that training continues on the multi-speaker SLR83 dataset, until convergence. In order to do that, when switching from pre-training to fine-tuning, a speaker lookup table is randomly initialized to condition the encoder on to-be-learned speaker representations. The intuition behind fine-tuning from a mono-speaker acoustic model is that of simplifying the overall training objective. In particular, in the pre-training stage, the model has to learn to correctly align text and audio and to produce meaningful phonetic representations for a single speaker, while in the fine-tuning stage the model has the task of generalizing what it learned in the previous phase to new speakers. Given the relatively low-resource setting of the SLR83 dataset, in terms of the amount of data for each speaker and dialect, we find this pre-training and fine-tuning strategy to be effective.

Table 5.5 and Figure 5.17 report results for the multi-speaker acoustic model pre-trained on LJ Speech and fine-tuned on SLR83 for $1.5\,\mathrm{M}$ steps. Both informal listening and objective metrics reveal that produced speech is intelligible and close to the speaker selected for synthesis, even though most voices tend to sound harsh and have major stability issues, such as word skipping. Such results were computed on 1000 samples, 100 from each of 10 randomly selected speakers. **Test speakers** are representative of each dialect since we select 1 for Irish and Welsh and 2 for each of the other dialects (Midlands, Southern, Northern and Scottish). We only select 1 speaker for Irish and Welsh as they are very under-represented in the dataset and we expect speakers in those dialects to perform poorly at inference time. Something worth pointing out is that the multi-speaker model beats the single-speaker ones introduced in Subsection 5.4.1 in terms of **intelligibility** metrics, such as WER and CER: we attribute this gap to the better stop token prediction

from the multi-speaker model, which avoids it producing utterances longer than required. For reference, centroids for cosine similarity are computed using 100 recordings for each speaker and the same utterances (1 per speaker) are used as reference values to compute MOS scores.



(a) Speaker similarity.

(b) MOS scores.

Figure 5.17: Tacotron 2 multi-speaker test score distributions on SLR83.

Speaker similarity (Cosine)	**Naturalness (MOS)**	**Intelligibility (WER)**	**Intelligibility (CER)**
26.33 ± 12.55	2.54 ± 0.23	65.88	42.18

Table 5.5: Multi-speaker acoustic modeling results on SLR83.

As we can observe, **cosine similarities** in Table 5.5 drop by a large margin w.r.t. the single-speaker results presented in Subsection 5.4.1. Nevertheless, informal listening reveals that the produced speech is highly correlated with the identity of the target speaker. Thus, we assume that other instabilities such as harshness and low naturalness may have an impact on the measured cosine similarity values. To counteract this problem, we hypothesise that speaker similarity could be perceptually improved by relying on **auxiliary losses**, such as the Speaker Consistency Loss (SCL) [193]. In the case of SCL, a pre-trained speaker encoder is used to extract speaker embeddings from the generated audio and ground truth, on which we maximize the cosine similarity. Formally, let ϕ be a function outputting the embedding of a speaker, s be the cosine similarity function, α a positive real number that controls the influence of the SCL in the final loss, and n the batch size. Then, SCL is defined as reported in Equation 5.3, where g and h represent,

respectively, the ground truth and the generated speaker audio.

$$L_{SCL} = -\frac{\alpha}{n}\sum_{i=1}^{n} s(\phi(g_i), \phi(h_i)) \tag{5.3}$$

Referring back to the pipeline introduced in Chapter 3, we also experimented with extending the multi-speaker architecture to model **multiple accents**. Unfortunately, multiple experiments aimed at reproducing results obtained with the multi-speaker model described above do not lead to the same winning path. In particular, what we observe is that the additional conditioning given by dialect accent embeddings makes the learning problem dramatically more complex, given the number of combinations between speakers and dialects the model has to cope with. Our main hypothesis is that the SLR83 corpus is simply not copious enough for Tacotron 2 to effectively deal with the additional variability introduced by dialects. To counteract this issue, we propose a **third pipeline** to model dialect accents in a TTS architecture, which should be more sample-efficient. Figure 5.18 outlines such pipeline, which assumes to work with a mono-speaker acoustic model (ideally speaking *neutral* British English, such as RP) and performs a joint speaker and accent conversion in a one-to-many fashion. In this way, the acoustic model only needs to learn to deal with a single speaker, and the conversion approach would be simplified by the one-to-many paradigm, since only the *neutral* speaker identity and accent would need to be converted to all other speakers, as opposed to the many-to-many approach where each voice can be converted to all the others available in the training set. Experiments with this simplified pipeline are left for future work.

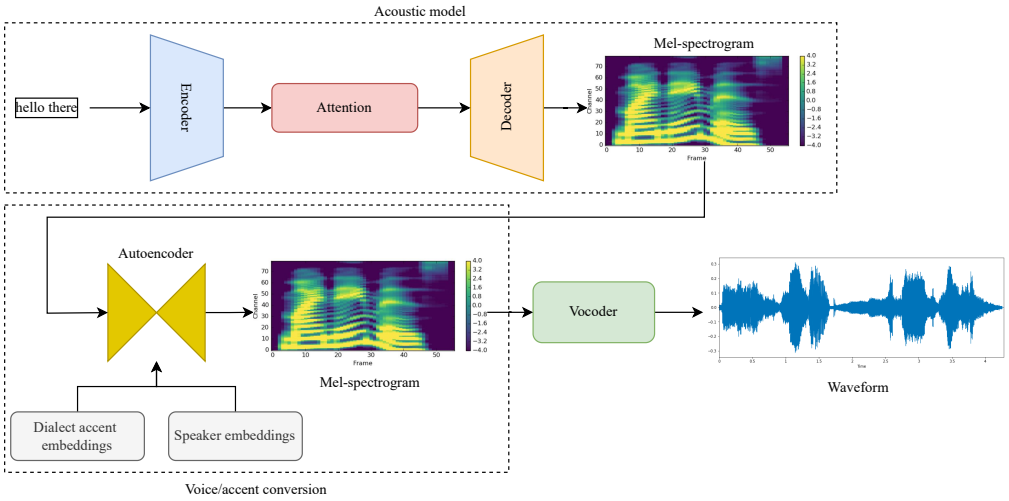


Figure 5.18: Acoustic model with neutral voice and a joint voice and accent conversion system.

5.5 Voice/accent conversion

Moving to the voice and accent conversion experiments, we first present the common training setup and results obtained with AutoVC on the VCTK and SLR83 datasets, when trained to carry out the standard VC task. As with acoustic modelling, this is done to assess the correctness of our custom implementations and presented in Subsection 5.5.1. Then, Subsection 5.5.2 deals with the adaptation of AutoVC to the AC case.

All AutoVC trainings share the **hyper-parameters** reported in Table 5.6, following [142]. For the VC task we always rely on the pre-trained Resemblyzer speaker embeddings presented in Section 3.1, while for the AC task we either make use of our custom 192-dimensional TitaNet dialect accent embeddings (trained on SLR83) or on to-be-trained lookup tables. For both voice and accent conversion, we follow the data splitting procedure reported in Section 4.2, after removing all silence regions from raw waveforms, resampling to 22 050 Hz and extracting mel-spectrograms. Moreover, during the batch creation phase, we rely on **random chunking** as a form of data augmentation and regularization technique. We also extract random chunks of random lengths (either 1.5 or 2 or 3 seconds), whenever the waveform's duration exceeds 3 seconds.

The following is a comprehensive **list of differences** between our workflow and that of AutoVC's authors[11]. In the data pre-processing stage, we do not limit mel-spectrogram frequencies (which authors clip in the 90 to 7600 Hz range) and do not rely on any DSP trick. Instead, authors first pass the raw signal through a 5th order Butterworth filter, to remove drifting noise, and then add back controlled noise to improve model robustness, following Equation 5.4, where w is the input waveform and r is a vector having the same shape as w and populated with random samples from a uniform distribution over $[0, 1)$.

$$0.96 \cdot w + (r - 0.5) \cdot 1 \times 10^{-6} \tag{5.4}$$

Moreover, authors normalize mel-spectrograms using the same reasoning explained in Section 4.2, but they apply an asymmetric normalization in the $[0, 1]$ range. Instead, we skip the normalization stage and directly use the log-magnitude of mel-spectrograms. Next, authors downsample training data (from the VCTK dataset in their case) to 16 kHz and randomly extract 128 spectrogram frames for each forward pass (which roughly correspond to 2 seconds of audio with their STFT settings), while we use a sampling rate of

[11] https://github.com/auspicious3000/autovc

22 050 kHz and rely on a different number of spectrogram frames at each forward pass (depending on the selected size of the random chunk), following [104], where authors recommend using 192 frames when the sampling rate is 22 050 Hz, in comparison with 128 frames for 16 000 Hz samples. Furthermore, authors suggest using a batch size of 2, due to the regularization effects of small mini-batches [156], but for most trainings, we decide to trade-off some generalization to speedup the learning phase and the overall experimentation pipeline and pivot to a larger batch size of 16 (the same setting used in [104]).

All training runs make use of the Adam optimizer [85] to minimize reconstruction and content losses. As in acoustic modelling, a single representation for each speaker and dialect accent is used as conditioning and that's either pre-computed as the centroid of all utterances for a certain speaker/accent, or learned through a lookup table at training time.

Before coming up with the **hyper-parameters** in Table 5.6, we experiment with multiple configurations. For example, we noticed that using a learning rate higher than 0.0001 (e.g. 0.001) leads to severe overfitting, whereby the model quickly learns how to reconstruct the input signal, but fails to correctly convert the desired speech attribute (e.g. speaker identity). Moreover, feeding entire spectrograms to the model makes it slightly overfit the training data, while relying on small chunks of audio seems to provide the right amount of regularization. The main parameters that were changed between different training runs are the bottleneck size and the downsampling factor of the encoder, which are essential to tune to obtain meaningful conversions. In the original paper, AutoVC's authors rely on a bottleneck size of 64 and a downsampling factor of 16.

5.5.1 Voice conversion

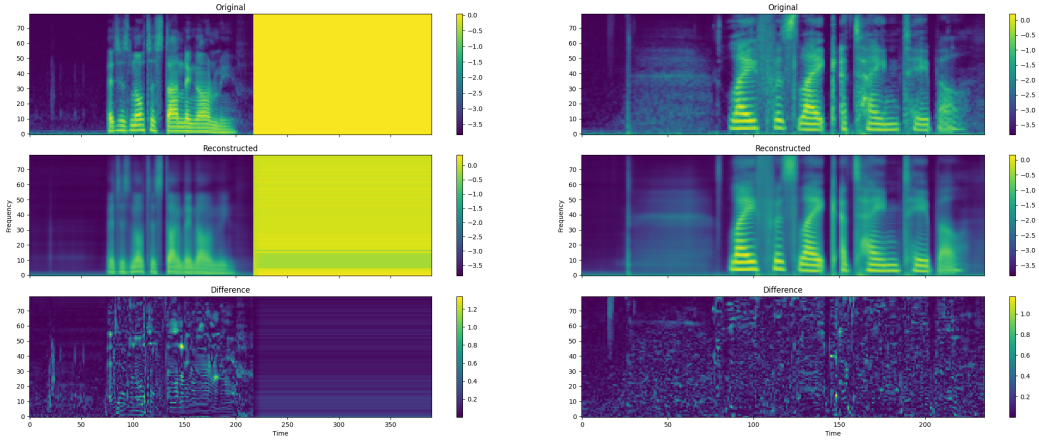
In voice conversion, the goal is to morph the voice of a source speaker to that of a target speaker, while maintaining linguistic content unchanged. To test our custom implementation of the voice conversion system, i.e. AutoVC, we first train a VC model on the **VCTK dataset**, which is the same multi-speaker corpus used in [142]. For this experiment, we rely on a subset of 40 speakers (out of 109 in total) from VCTK, the standard bottleneck size of 64 and a more aggressive downsampling factor of 32 (w.r.t. the default one of 16). Moreover, this training run was performed with the batch size of 2 suggested by the authors, which required about 4 days of training on our modest hardware setting.

Parameter	Value
Batch size	16
Learning rate	0.0001
Gradient norm clip value	2.0
Encoder number of convolutional layers	3
Encoder number of convolutional filters	512
Encoder convolutions kernel size	5
Encoder recurrent unit	LSTM
Encoder number of recurrent layers	2
Decoder number of convolutional layers	3
Decoder number of convolutional filters	512
Decoder convolutions kernel size	5
Decoder recurrent unit	LSTM
Decoder number of recurrent layers	3
Decoder recurrent bi-directional	False
Decoder recurrent hidden size	1024
PostNet hidden size	512
PostNet number of convolutional layers	5
PostNet convolutions kernel size	5
Reconstruction loss weight	1.0
Content loss weight	1.0

Table 5.6: AutoVC hyper-parameters.

Convergence was observed at the 40 epochs mark, which equals around 65 K training steps. In [142], the suggested amount of steps to train for in the same settings is 100 K, even though we find it sufficient to train for less to obtain a reasonable conversion quality.

Figure 5.19 reports the **progress in self-reconstruction** during training. In particular, Figure 5.19a shows that at 10 K steps the model has barely learned to recover high-level details of the given mel-spectrograms, while in Figure 5.19b it can be observed that fine-grained features can be reconstructed as well. Beware that the 2 reported examples in figure 5.19 depict different utterances not because of cherry-picking but because of both random chunking and the fact that at each training epoch a random utterance was selected for visualization. Something else worth pointing out is that this specific training run was carried out with no VAD pre-processing whatsoever, due to a bug in the data workflow, that was fixed after analysing the results of this experiment. Still, we find that even without silence removal, we are able to successfully converge on the VCTK dataset.



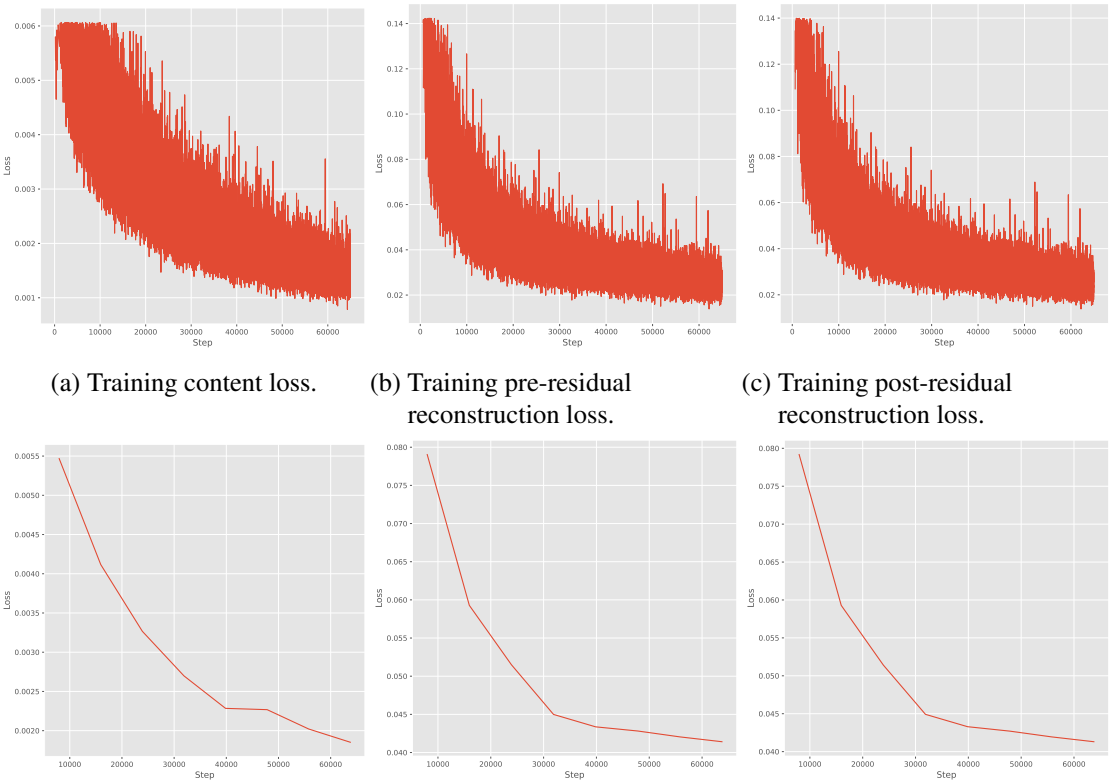
(a) At 10 K steps.

(b) At 65 K steps.

Figure 5.19: Self-reconstruction quality of AutoVC over training on VCTK. The model quickly learns to reconstruct target spectrograms, but slowly adapts to fine-grained details.

Instead, Figure 5.20 shows the training and validation losses throughout the learning phase. As we can see, we are able to reach convergence with a reconstruction loss of 0.02, instead of the 0.0001 value suggested by the authors.

Before settling for 40 as the number of target speakers, we experimented with different values, such as using **all speakers**, but obtained comparatively worse conversion results. This may seem counter-intuitive from the point of view of DL, given that we expect more data to help produce better models, but we have to keep in mind that stronger



(a) Training content loss.

(b) Training pre-residual reconstruction loss.

(c) Training post-residual reconstruction loss.

(d) Validation content loss.

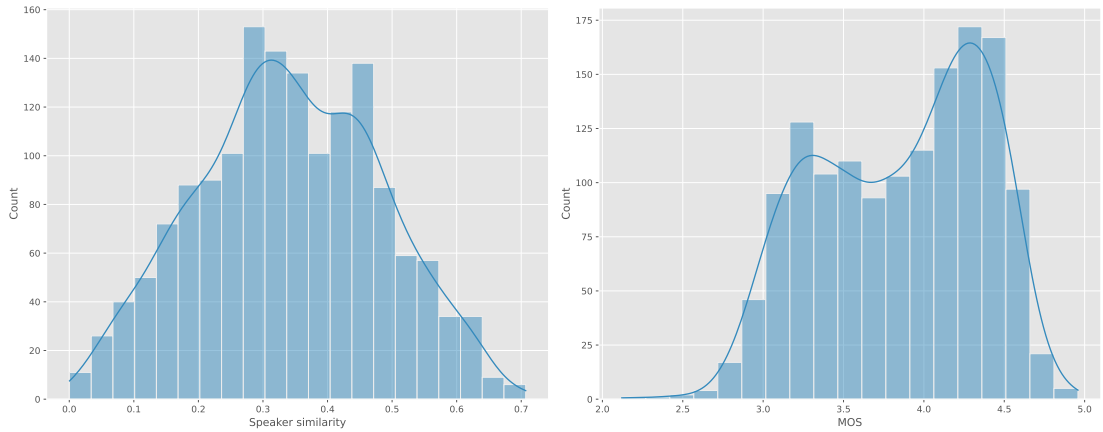
(e) Validation pre-residual reconstruction loss.

(f) Validation post-residual reconstruction loss.

Figure 5.20: AutoVC training and validation losses on VCTK.

generalization usually comes at the expense of longer training time. Hence, we hypothesise that training for longer with all speakers should yield the same results, in terms of conversion quality, than training for less with a subset of speakers.

To better understand the **generalization ability** of our VC model trained on VCTK, we extract 1 text prompt for each of the 40 speakers in the training set and convert each prompt to all the other 39 speakers, thus obtaining a total of 1560 utterances (39 for each target speaker). Please note that even though text prompts and speakers were seen during training, each combination of source prompt and target speaker was instead unseen; thus, the test set we rely on is valid for seen-to-seen conversion. After building the test set and having converted all utterances within, we run evaluations as explained in Section 5.2 and report distributions for speaker similarity and MOS scores in Figure 5.21, along with naturalness and intelligibility metrics in Table 5.7 and Table 5.8. In this evaluation, the centroid of a speaker is computed as the average embedding over 100 randomly selected ground-truth utterances. Analysing results in Table 5.7, we can see that WER and CER metrics are not impressive, but, in order to get a sense of such intelligibility metrics we provide upper bounds, by computing them on a subset of ground-truth utterances, referred to as *"Reference"* in Table 5.8. This is done by taking 1 prompt for each speaker (for a total of 40 inputs), which gives us a WER of 37.31 and a CER of 9.67, meaning that our VC system tends to lose around 30 percentage points for both intelligibility metrics w.r.t. recordings. Beware that Table 5.7 only reports results for the test-set and not for ground truth.



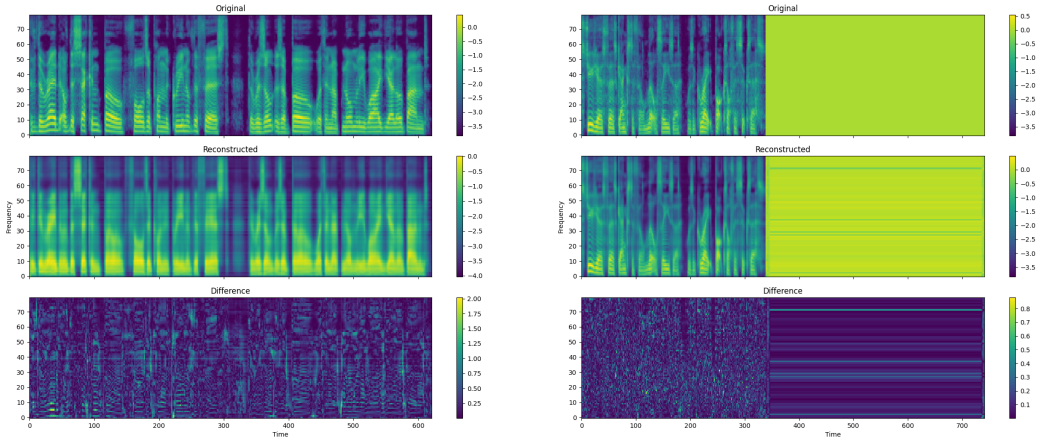
(a) Speaker similarity over the entire test set.

(b) MOS scores over the entire test set.

Figure 5.21: AutoVC voice conversion test score distributions on VCTK.

Moving to the **SLR83 dataset**, the best results were obtained when sticking to the

same settings used with VCTK. In particular, 40 speakers were selected (without balancing based on dialects) and a bottleneck size of 64 was used along with a downsampling factor of 32. As with VCTK, we observe convergence when the training reconstruction loss reaches the 0.02 mark, which happens after 110 epochs. Figure 5.22 shows examples of mel-spectrogram self-reconstructions on the validation set at the start and towards the end of training.



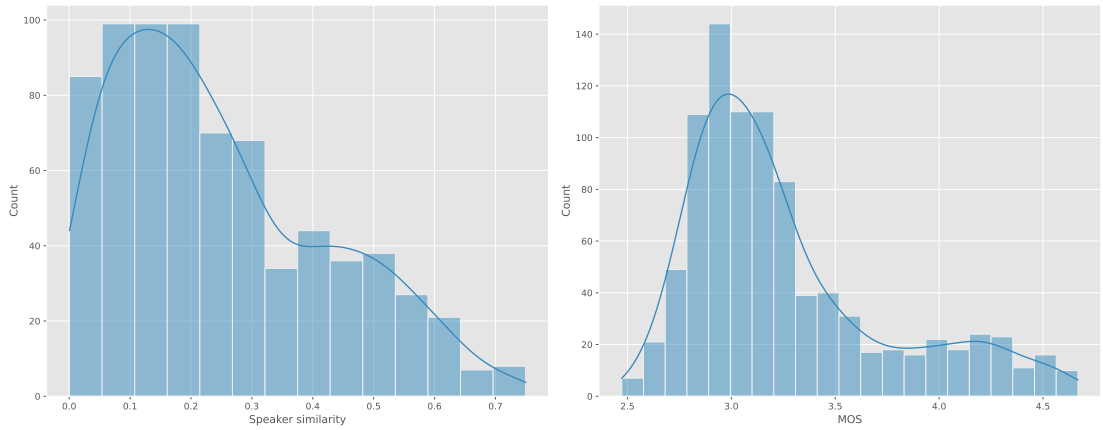
(a) At 1 K steps. (b) At 30 K steps.

Figure 5.22: Self-reconstruction quality of AutoVC over training on SLR83.

In this experiment, we rely on a batch size of 16 to speed training up and we clip the norm of the gradient to a value of 2.0, instead of 1.0. Even though we were able to use a bigger batch size (and still observe convergence), training still required more than 4 days. Something worth pointing out is that this experiment was executed with both speaker and dialect accent conditioning, but we only evaluate **conversion of the speaker identity** in this section and leave accent conversion evaluations for Subsection 5.5.2. In particular, at inference time we make sure to only convert speaker information by conditioning AutoVC on the target speaker and the source accent. It's worth noticing that the test set includes dialect accents unseen during training and the model is still able to perform a successful identity conversion.

To build the SLR83 test set we extract 3 speakers for each dialect (i.e. 18 speakers in total) and 3 utterances for each speaker (for a total of 54 utterances). Then, each utterance is converted from the source speaker to all the target speakers, while keeping the source dialect fixed, to get 918 utterances to run evaluations on. Results are still reported in Table 5.7 and Table 5.8, with Figure 5.23 depicting test score distributions.

Looking closely at results in Table 5.7 and Table 5.8, we see that all metrics are



(a) Speaker similarity over the entire test set.

(b) MOS scores over the entire test set.

Figure 5.23: AutoVC voice conversion test score distributions on SLR83.

comparable with VCTK results, even though they all lag a bit behind. As done for VCTK, we compute WER and CER metrics on a subset of ground-truth utterances (1 for each of the 17 speakers in the test-set) to obtain values of 26.97 and 8.86, respectively, meaning that the VC process hinders intelligibility by around 50 percentage points (around 20 more than on VCTK). Something worth noticing is that, even though conversion results are perceptually of lower quality w.r.t. the ones produced on VCTK, they are not as bad as they might seem just by looking at the objective metrics. Indeed, what we observe is that for speech systems the gap between such measures and human perception is very high and little work has been done to perform automatic evaluations of these systems.

	Speaker similarity (Cosine)	**Naturalness (MOS)**	**Intelligibility (WER)**	**Intelligibility (CER)**
VCTK	34.27 ± 13.96	3.85 ± 0.52	70.82	39.10
SLR83	24.82 ± 17.61	3.26 ± 0.48	77.28	50.30

Table 5.7: AutoVC voice conversion results.

5.5.2 Accent conversion

This section deals with extending the AutoVC architecture to the task of **regional accent conversion**. To do so, we condition AutoVC's encoder not only on speaker representations but also on dialect accent ones. Here, the intuition is that the model should discard those pieces of information that are provided as input from the bottleneck features, thus

		VCTK		SLR83	
		Reference	Test	Reference	Test
WER	Substitutions	97	6291	51	5981
	Insertions	4	647	3	972
	Deletions	2	769	4	2822
	Hits	177	3821	160	3845
	Incorrect	103	7707	58	9775
	Total	276	10 881	215	12 648
CER	Substitutions	2840	12 879	1292	15 206
	Insertions	440	3470	170	3865
	Deletions	2320	5778	374	17 956
	Hits	52 720	37 932	19 040	40 448
	Incorrect	5600	22 127	1836	37 027
	Total	57 880	56 589	20 706	73 610

Table 5.8: AutoVC voice conversion intelligibility details, in terms of the number of insertions, deletions and substitutions carried out.

enabling text, speaker and accent **disentanglement**. As with VC, tuning the bottleneck is one of the most important tasks to carry out in order to obtain meaningful conversion.

We experiment with 2 ways to perform AC: the first one is by conditioning the model on both speaker and dialect accent embeddings, and the second one is to only condition the model on dialect accent embeddings. For the **first approach**, we are able to convert speaker s from its native dialect d_s to the target dialect d_t by conditioning the model on the same source speaker s and on the target dialect d_t. Instead, for the **second approach** we simply condition the model on the target dialect d_t. In theory, the first approach could also be used to perform joint voice and accent conversion, but we don't test this scenario as it's considered out of scope for this thesis.

Informal evaluations reveal that there is a strict **trade-off** between speaker and accent conversion, meaning that successful conversion of accent usually comes at the cost of substantial **speaker leakage**. What we observe is that conditioning the model on accent alone makes it utter sound that's more similar to the target accent, but the original speaker identity gets mixed with what we believe are the identities of other speakers in the same dialect, which results in artefacts such as unpleasant mid-sentence pitch shifting. Instead, in our experiments, conditioning the model on both speaker and dialect accent information leads to more consistent speaker similarities, but weaker accent conversion

capabilities.

For the first approach, we rely on the same parameters used in Subsection 5.5.1, i.e. a bottleneck size of 64 and a downsampling factor of 32, while for the second approach we experiment with multiple hyper-parameters and identify **smaller hidden sizes** to work much better than the ones reported so far. In particular, when the model is only conditioned on dialect accent we find that relying on small 8-dimensional lookup tables, 4 as the bottleneck size and 2 as the downsampling factor results in an overall better conversion.

Due to the severe speaker leakage observed in the accent-only conditioning scenario, we only report results for the model where both target speaker and accent are controllable. In particular, Table 5.9 shows MOS results for accentedness obtained from a crowd-sourcing evaluation on Mechanical Turk, while Figure 5.24 and Table 5.11 outline speaker similarity, naturalness and intelligibility, as done with systems in the previous sections. Given that the model was trained on a **subset of speakers**, we only evaluate those dialects for which at least one speaker was present in the training set, i.e. all but Southern and Welsh.

Evaluations for **accentedness** are carried out in the following way. First, we select 1 random speaker for each included dialect and sample 6 utterances from the SLR83 dataset for each speaker, resulting in 24 utterances in total. Then, we convert each utterance from the source dialect to all others and further perform source-to-source dialect conversion for 2 utterances in each dialect[12], thus obtaining 80 utterances, that we ask workers in the crowd-sourcing platform to evaluate by only judging accent and ignoring other dimensions such as naturalness and intelligibility. Results are filtered based on a single criterion, which is that the **task duration** (i.e. the amount of time the worker spent on evaluating a sample) should be strictly higher than the audio duration: in case the task duration is lower than the audio duration, the worker either didn't listen to the sample in its full length or randomly guessed one of the available answers. Results were processed by 33 unique workers and the total number of submissions is 800 (10 for each sample). The last row in Table 5.9 represent the mean MOS (with *micro* and not *macro* aggregation) for each target dialect, while the last column represents the same but for source dialects. Moreover, the bottom right cell of the table contains an **overall score** for the

[12]The source-to-source dialect conversion is performed to have an upper bound on MOS scores. We could've used recordings in place of source-to-source conversions, but we decide to adopt a strategy similar to copy-synthesis for the evaluation of vocoders.

system, still computed with a *micro* strategy[13]. As we can see, the **best source-to-target conversion** is obtained from Northern to Midlands, while the second place is taken by the Scottish to Midlands pair, which we actually consider to be more noteworthy than the former given the greater linguistic distance. Unexpectedly, the Northern to Midlands conversion achieves better results than the Midlands to Midlands one, meaning that when the speaker with Northern accent is converted to have an accent from the Midlands, it actually sounds more native than a native Midlands speaker. Notice that this is true for the test case we selected, but it might be that a different pool of speakers would yield different results, depending on how strong their accent is.

	Converted dialect				
Original dialect	**Irish**	**Midlands**	**Northern**	**Scottish**	**Average**
Irish	3.56 ± 1.10	2.87 ± 1.36	2.80 ± 1.18	3.18 ± 0.98	2.96 ± 1.17
Midlands	2.10 ± 1.32	3.60 ± 1.07	3.55 ± 1.15	2.85 ± 1.39	2.73 ± 1.41
Northern	2.98 ± 1.37	3.63 ± 0.98	3.70 ± 1.15	2.50 ± 1.65	3.21 ± 1.29
Scottish	3.18 ± 1.26	3.40 ± 1.22	2.96 ± 1.37	4.38 ± 0.94	3.15 ± 1.29
Average	2.66 ± 1.32	2.95 ± 1.27	2.85 ± 1.27	2.61 ± 1.14	2.77 ± 1.26

Table 5.9: AutoVC accent conversion crowd-sourced accentedness. The bottom-right cell indicates the overall score for the system.

To further evaluate the **statistical significance** of results in Table 5.10, we evaluate **inter-rater reliability** using the Cronbach's alpha measure [35], which intuitively encodes how similar is the internal rating system of different listeners[14] [113]. Other than inter-rater reliability, another important metric to observe would be the **intra-rater reliability**, i.e. how consistent each listener is in using the 1 to 5 discrete rating system. Unfortunately, inter-rater reliability metrics usually rely on double ratings, whereby the same listener evaluates the same sample twice[15], which we decide to exclude for budget limitations. Thus, we only evaluate the inter-rater reliability and report results in Table 5.10. As we can see, the **overall reliability** exceeds 0.95 for the Cronbach's alpha value[16], meaning that another sample of listeners would produce the same (or a very

[13] A macro-average computes the metric independently for each class and then takes the average, whereas a micro average aggregates the contributions of all classes to compute the average metric.

[14] All listeners must rate a given speech sample in a similar way.

[15] A given listener must rate a specific speech sample the same way every time he or she hears it.

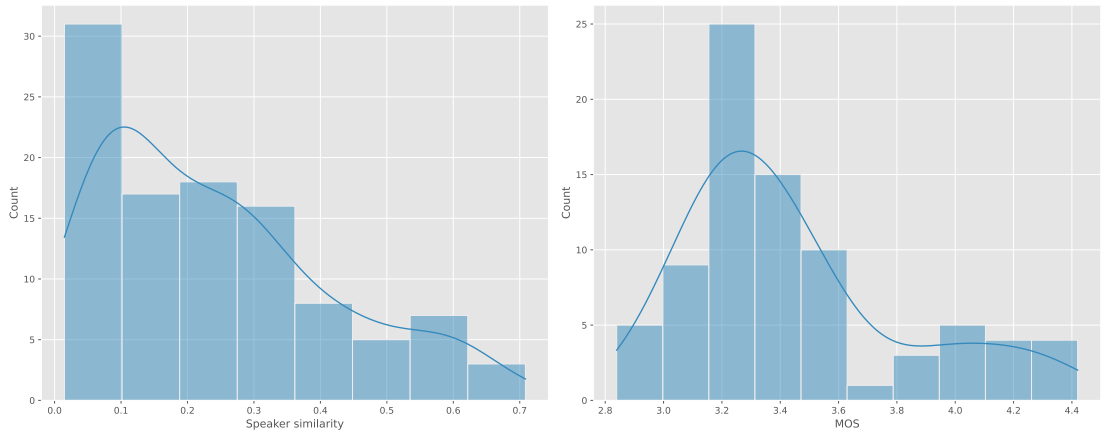
[16] For Cronbach's alpha, internal consistency ranges between negative infinity and one. Also, the coefficient alpha will be negative whenever there is greater within-subject variability than between-subject variability.

similar) mean rating score for the same speech material.

Converted dialect	Cronbach's alpha	CI 95%
Irish	0.88	[0.80, 0.94]
Midlands	0.81	[0.69, 0.90]
Northern	0.88	[0.69, 0.93]
Scottish	0.93	[0.88, 0.96]
Average	0.96	[0.94, 0.98]

Table 5.10: AutoVC accent conversion crowd-sourced inter-rater reliability, as measured by Cronbach's alpha [35].

Finally, Table 5.11 and Figure 5.24 report measurements in terms of speaker similarity, naturalness and intelligibility. As we can see, results are comparable to the VC ones presented in Subsection 5.5.1. In particular, we observe an **improvement in intelligibility**, where scores only drop by 25 to 30 percentage points compared to ground-truth references, and a slight **decrease in speaker similarity**, which is expected given the more complex task of accent conversion.



(a) Speaker similarity. (b) MOS scores.

Figure 5.24: AutoVC accent conversion test score distributions on SLR83.

Speaker similarity (Cosine)	Naturalness (MOS)	Intelligibility (WER)	Intelligibility (CER)
24.07 ± 17.22	3.44 ± 0.38	56.18	30.92

Table 5.11: AutoVC accent conversion results on SLR83.

5.6 Joint TTS and voice/accent conversion

In this section, we evaluate results on our proposed pipeline composed of a multi-speaker acoustic model and followed by an AC model, to enable full controllability of speaker and dialect accent.

The first evaluation consists in **stacking** the multi-speaker acoustic model of Subsection 5.4.2, trained on SLR83, with the AC system of Subsection 5.5.2. What we observe is that combining the two models leads to unintelligible speech being produced by the whole pipeline. We attribute such unsatisfactory results to the **propagation of errors** of each component to the ones at the following stages. In particular, the harshness of sound produced by the acoustic model makes the AC system fail due to reduced accent similarity between ground-truth and synthesised speech. In turn, the vocoder outputs a very noisy signal, given the unstable mel-spectrogram predictions by the previous modules. Thus, we hypothesise that more training iterations or other forms of careful fine-tuning of each component would lead to valid outputs.

The second evaluation is instead related to the **end-to-end approach** presented in Section 3.5. For this experiment we rely on the YourTTS checkpoint available in the authors' GitHub repository[17], which is a VITS model pre-trained for $1\,\mathrm{M}$ steps on LJ Speech and then fine-tuned gradually in the following way. First, training is resumed for $200\,\mathrm{K}$ steps with the VCTK dataset to adapt the single-speaker system to a multi-speaker one. Then, training continues for approximately $140\,\mathrm{K}$ steps on VCTK, TTS-Portuguese[18] [18] and M-AILABS [57] French[19] datasets, in order to make the model tri-lingual. Finally, the last training stage adds 1151 additional English speakers from both LibriTTS partitions *train-clean-100* and *train-clean-360*.

To adapt YourTTS to the task of **accent conversion**, we consider each dialect as a distinct language and fine-tune the checkpoint described above with 6 additional dialects. In terms of **hyper-parameters**, we adopt the default ones provided by the Coqui TTS repository. In particular, we fine-tune for $500\,\mathrm{K}$ steps using a batch size of 8, after resampling raw waveforms to $16\,\mathrm{kHz}$ (instead of the $22\,050\,\mathrm{Hz}$ used so far), trimming silences and normalizing mel-spectrograms to the $[-4, 4]$ symmetric range. On the target text side, we

[17] https://github.com/edresson/yourtts

[18] The TTS-Portuguese corpus [18] is a single-speaker dataset of the Brazilian Portuguese language with around 10 hours of speech, sampled at $48\,\mathrm{KHz}$.

[19] M-AILABS [57] is a dataset based on LibriVox [108] and its French subset consists of 2 female ($104\,\mathrm{h}$) and 3 male speakers ($71\,\mathrm{h}$) sampled at $16\,\mathrm{KHz}$.

also remove the beginning and end of sentence tokens. Such pre-processing steps were necessary to comply with the settings used in the pre-training stage. The model is fine-tuned with the AdamW optimizer [115] with $\beta_1 = 0.8$, $\beta_2 = 0.99$ and $\epsilon = 1 \times 10^{-9}$, using a weight decay of 0.01 and an initial learning rate of 0.0002 that's annealed exponentially by a gamma of 0.999875. Regarding the speaker information, we rely on one-hot encodings of speaker indices, without any additional loss term to enforce speaker consistency. As with speaker embeddings, dialect representations are learned using a 6×4 embedding matrix, where 6 is the number of dialects and 4 is the dimensionality of the latent space they're mapped to. No layer is frozen and they are all fine-tuned on the SLR83 dataset. Moreover, we rely on weighted random sampling to guarantee a dialect-balanced batch, as in [20].

Given the great results obtained with this end-to-end approach, we decide to run **formal evaluations** and compare them with what we obtained in Subsection 5.5.2 for accent conversion through AutoVC. In particular, Table 5.12 reports by-dialect means and standard deviations for the same **crowd-sourced accentedness** test performed in Subsection 5.5.2. Differently from AutoVC, YourTTS is trained on the full set of dialects and we thus report results for each of them, but only compare systems on the shared ones. For the crowd-sourced tests, we select a super-set of the prompts used to evaluate AutoVC, by selecting a total of 120 utterances: 18 per target dialect plus 12 for source-to-source conversion. Then, we run a Mechanical Turk test and filter submissions based on the same criteria used in Subsection 5.5.2: out of 1200 submissions carried out by 36 unique workers, we only exclude 7 submissions that we identify as random guesses.

This test reveals the same pattern observed in Subsection 5.5.2, whereby for some dialects the **source-to-target conversion** (where the source and target dialects are different) achieves greater scores than the source-to-source one. For example, we see that Midlands to Irish seems better than Irish to Irish; Irish to Northern is better than Northern to Northern; Northern to Scottish is better than Scottish to Scottish; Northern to Southern, Welsh to Southern and Scottish to Southern are better than Southern to Southern; and Scottish to Welsh and Irish to Welsh are better than Welsh to Welsh. These patterns could be due to the way the model has learned different dialects (which mirrors how under or over-represented each class is in the SLR83 dataset) and to the number of evaluations for each sample.

As done in Subsection 5.5.2, we compute the **Cronbach's alpha** metric as a proxy

Original dialect	Converted dialect: Irish	Midlands	Northern	Scottish	Southern	Welsh	Average
Irish	3.56 ± 1.10	3.43 ± 1.19	3.59 ± 1.14	3.88 ± 1.22	3.38 ± 1.21	3.60 ± 1.01	3.58 ± 1.16
Midlands	4.37 ± 0.72	3.80 ± 1.32	3.35 ± 1.42	3.70 ± 1.08	3.40 ± 1.47	3.23 ± 1.43	3.64 ± 1.30
Northern	3.93 ± 1.25	3.50 ± 1.20	3.33 ± 1.35	4.30 ± 1.06	4.00 ± 0.78	3.04 ± 1.48	3.70 ± 1.22
Scottish	3.88 ± 1.02	3.53 ± 1.04	3.10 ± 1.18	4.20 ± 0.83	3.63 ± 1.09	3.98 ± 0.89	3.62 ± 1.10
Southern	3.50 ± 1.00	3.22 ± 1.25	3.25 ± 0.91	4.12 ± 0.88	3.60 ± 1.17	3.38 ± 1.15	3.57 ± 1.12
Welsh	3.83 ± 1.26	2.76 ± 1.02	3.08 ± 1.26	4.08 ± 0.99	4.00 ± 1.08	3.40 ± 0.99	3.55 ± 1.24
Average	3.92 ± 1.10	3.30 ± 1.18	3.25 ± 1.20	4.02 ± 1.03	3.67 ± 1.12	3.48 ± 1.21	3.60 ± 1.18

Table 5.12: YourTTS accent conversion crowd-sourced accentedness. The bottom-right cell indicates the overall score for the system.

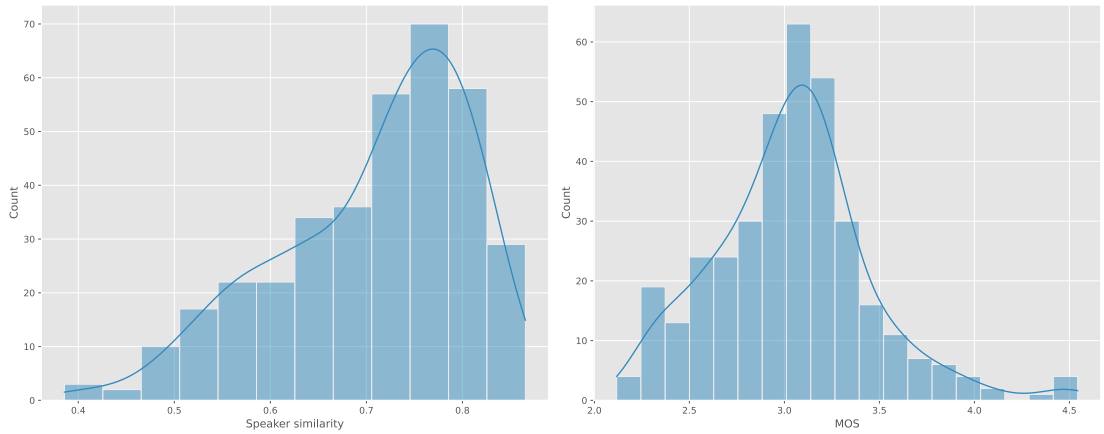
measure for inter-rater reliability and report results in Table 5.13. As we can see, there is a strong confidence that repeating the same test with different listeners would yield very similar results for all dialects but the Scottish one, where the degree of uncertainty is relatively high.

Converted dialect	Cronbach's alpha	CI 95%
Irish	0.85	$[0.75, 0.92]$
Midlands	0.95	$[0.93, 0.97]$
Northern	0.92	$[0.87, 0.95]$
Scottish	0.42	$[0.07, 0.68]$
Southern	0.91	$[0.85, 0.95]$
Welsh	0.77	$[0.62, 0.88]$
Average	0.95	$[0.92, 0.97]$

Table 5.13: YourTTS accent conversion crowd-sourced inter-rater reliability, as measured by Cronbach's alpha [35].

Finally, we report results in terms of **speaker similarity, naturalness and intelligibility** in Table 5.14 and Figure 5.25. As we can see, intelligibility improves by a large margin w.r.t. AutoVC conversions, with YourTTS only losing about 10 and 5 percentage points, respectively for WER and CER, when compared to recordings. The same trend can be observed in speaker similarity, with YourTTS achieving around 50 percentage points more than AutoVC in terms of cosine similarity. As discussed above, the relationship between cosine similarity and perceived speaker similarity does not seem to follow a linear trend, in that informally comparing results obtained with AutoVC and YourTTS

does not show gaps as big as the ones revealed by objective metrics, even though we can generally regard voices produced by YourTTS as of higher quality and more stable than the ones generated by AutoVC.



(a) Speaker similarity. (b) MOS scores.

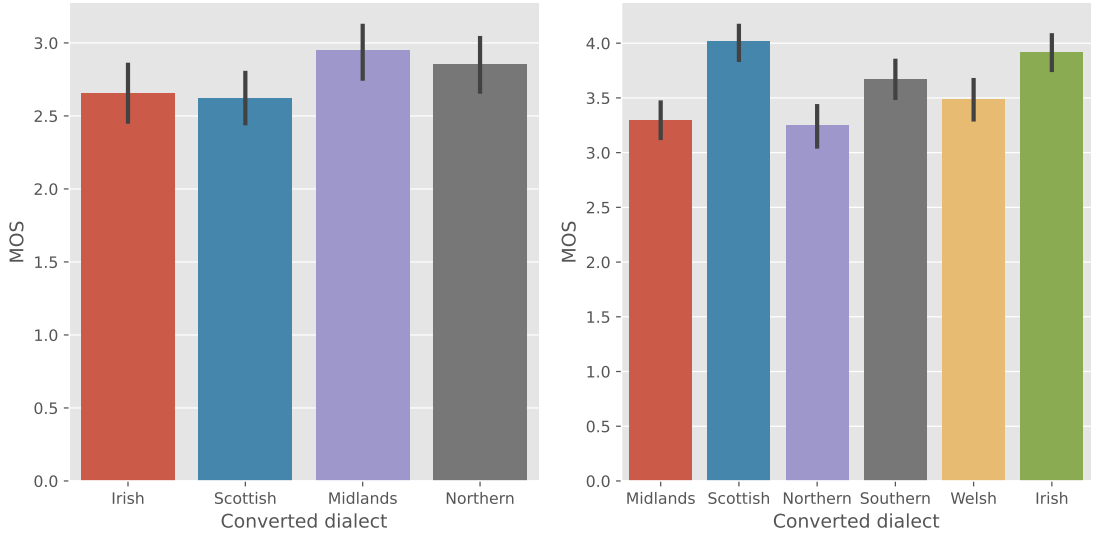
Figure 5.25: YourTTS accent conversion test scores on SLR83.

Speaker similarity (Cosine)	Naturalness (MOS)	Intelligibility (WER)	Intelligibility (CER)
70.70 ± 9.98	3.04 ± 0.41	38.95	16.62

Table 5.14: YourTTS accent conversion results on SLR83.

As a last piece of analysis, we perform **statistical tests** to assess the improvements obtained with **YourTTS w.r.t. AutoVC**, in terms of accentedness. To do so, we first give a visual contrast of the solutions in Figure 5.26 and then present inferences about the mean ratings obtained by the two systems in Table 5.15. To give an overall comparison, we first run an independent two-sample t-test and correct for the unequal sample sizes (given by the different number of dialects included in AutoVC and YourTTS trainings) using the Welch-Satterthwaite equation [150, 188]. Then, we run the same tests without Welch's correction on MOS scores for each dialect the two systems have in common. In all tests, we rely on the **null hypothesis** that the means are equal and the alternate hypothesis that they are different, i.e. the t-values and p-values are computed w.r.t. two-sided tests. Given that the mean scores for YourTTS in Table 5.12 are higher than scores for AutoVC in Table 5.9 and that the difference between such scores is statistically significant for all dialects, i.e. all t-tests have a p-value much smaller than the significance

level we set $\alpha = 0.05$, we conclude that YourTTS is better suited for the accent conversion task. Moreover, YourTTS beats AutoVC even when considering more dialects (i.e. Southern and Welsh) than the ones selected for training the latter. Finally, it has to be noted that YourTTS does not simply perform the accent conversion task, but acts as an **end-to-end system** to transduce text into speech with the desired speaker and accent attributes, which make results even more interesting given the more complex problem it has to face.



(a) MOS for AutoVC.

(b) MOS for YourTTS.

Figure 5.26: YourTTS vs AutoVC crowd-sourced accentedness on SLR83.

Converted dialect	t**-value**	p**-value**	**CI 95%**
Irish	9.85	2.02×10^{-20}	$[1.01, 1.51]$
Midlands	12.14	1.32×10^{-28}	$[1.17, 1.63]$
Northern	2.68	7.67×10^{-3}	$[0.09, 0.6]$
Scottish	3.03	2.65×10^{-3}	$[0.14, 0.65]$
Average	14.17	7.64×10^{-43}	$[0.72, 0.95]$

Table 5.15: YourTTS vs AutoVC accent conversion t-tests on SLR83. All tests are statistically significant, with p-values $\ll 0.05$.

5.7 Discussion of results

The main goal of this thesis has been to understand whether or not modern TTS systems have enough capacity to be conditioned on **regional accents**. In this way, synthesised speech may be controlled on variations of a language as subtle as the within-country ones, rather than modelling their between-country counterparts, as it's been done so far [126, 129, 200]. To answer this question, we devise 2 TTS **pipelines**, based on the popular Tacotron 2 [152] architecture, that take as input a piece of text and output a raw waveform with the desired speaker identity and dialect accent. We also compare the intermediate outputs of such pipelines with an adaptation of the YourTTS [20] multi-lingual model to the multi-dialect case.

The curse of multi-stage approaches Results show that each individual component is able to accomplish its target task when evaluated in isolation, but the combination of modules in an end-to-end pipeline fails to deliver. This is what we believe to be one of the greatest drawbacks of **multi-stage approaches**, in which the final output is produced by a sequence of steps where each step is dependent on the previous ones. In this way, in case one of the modules is not trained enough to closely resemble ground-truths, the model following it would have a hard time carrying out its specific task, given the **noisy input**. This, in turn, would have major negative consequences for the final output, since it would be the result of an accumulation of errors. Standard TTS pipelines already suffer from this issue, having at least 2 modules to be trained disjointly, i.e. the acoustic model and the vocoder. In this work, the number of **independent components** amounts to as much as 5 models, i.e. the acoustic model, the speaker and dialect accent encoders, the vocoder and the VC/AC systems. Thus, we hypothesise that reducing the number of independent components would be very beneficial for the final output. For example, in case zero-shot synthesis is not a requirement, one could rely on lookup tables for speaker/accent embeddings instead of having one encoder for each speech attribute. This hypothesis is also backed by the far better performance of YourTTS w.r.t. our proposed pipelines, which could be explained by the end-to-end nature of the model (other than its more extensive pre-training phase), where a single architecture is trained for acoustic modelling, vocoding and speaker/accent conditioning. We also propose a **third pipeline**, not implemented in this work, where a single-speaker acoustic model is followed by a more complex VC/AC system that carries out a one-to-many conversion task, from the

single *neutral* speaker to all the other speaker and accents in the dataset. Obviously, this shifts the burden from the acoustic model to the conversion system, but we believe this is a fair move, given that the latter tends to be easier and faster to train than the former module.

Speaker and dialect accents embeddings Moving to the specifics of **speaker and dialect accent embedding** models, we compare the well-established d-vector architecture with the SOTA TitaNet model. Results show that the true strength of the d-vector model comes from both its GE2E loss function and the size of the dataset used to train it. In particular, we show that the pre-trained **Resemblyzer** speaker embeddings generalize far better than our d-vector model, to both unseen utterances and unseen speakers, with the only differences between the two being in the use of a GE2E loss (instead of standard CE) and the scale of the training corpora. Instead, **TitaNet** generalizes well even when trained with simple loss functions and a limited set of utterances, even though Resemblyzer tends to be more consistent across multiple tests. For the dialect accent encoder, we only report results for the TitaNet model, given that no previous work could be found for this classification task on the selected dataset. Our evaluations reveal that TitaNet is able to correctly cluster together utterances belonging to the same dialect, even though some speaker leakage is observed. We believe that better **disentanglement properties** between speaker information and dialect accents could be obtained by relying on larger-scale corpora or explicit training strategies such as adversarial losses.

About the acoustic model For what concerns the **acoustic model**, we first show that we are able to obtain intelligible and natural speech in a single-speaker scenario and then extend the model to work in a multi-speaker setting, by conditioning it on the speaker embeddings extracted via Resemblyzer. Informal listening and subjective evaluations reveal that we are also able to model the SLR83 dataset while controlling synthesised speech for speaker identity. Instead, adding **dialect accent conditioning** on the same acoustic model makes the number of combinations of sounds to learn too large to be extensively covered by the SLR83 dataset, since it only has $30\,\mathrm{h}$ of speech.

Scale of the SLR83 dataset The SLR83 dataset suffers from severe **imbalance** on the duration of recordings both speaker-wise and dialect-wise, with certain dialects such as

Welsh and Irish being drastically under-represented. Moreover, we observed various issues with recordings, such as mispronunciations and word skipping, which do not help our attention-based TTS systems. We believe that relying on a **large-scale** curated corpora, such as the ones in the ABI family, would lead to more satisfactory results. Specifically, we hypothesise that a dataset balanced in the number of speakers per dialect and the duration of audio per speaker is key to learning meaningful representations.

Accent conversion with AutoVC On the VC/AC side of things, we first reproduce results in the AutoVC paper, to **convert speaker identity** on a subset of 40 VCTK speakers and repeat the same experiment on a subset of SLR83 speakers. Results show successful conversion for both corpora, even though the objective metrics we measured do not seem comparable to the ones reported in [142]. For what concerns **AC**, we show that AutoVC can carry out this task for a subset of 4 dialects, even though, for some utterances, converted speech either suffers from substantial speaker leakage or it does not sound like the target accent. Other than that, we also observe that AutoVC tends to **deteriorate linguistic content**, at least when its bottleneck is not properly tuned. We believe that more SOTA approaches, such as VoiceMixer [104] or StarGANv2-VC [107], would largely ameliorate such issues. Still, AutoVC achieves an overall MOS score of 2.77 ± 1.26 for **perceptual accentedness** on the AC task, with conversions to Midlands and conversion from Northern accents being the higher scoring ones, with MOS means of 2.95 and 3.21, respectively. We want to remind the reader that an MOS score of 1 means that the speaker does not have the evaluated accent at all, while a score of 5 equates to a completely native accent.

Accent conversion with YourTTS Finally, we evaluate YourTTS on the **end-to-end** task of TTS synthesis with fine-grained control of speaker identity and dialect accent. In particular, we **fine-tune** YourTTS, pre-trained on a multi-lingual dataset, with the multi-dialect SLR83 corpus and show that the model can adapt to variations of British English at a regional level. In particular, we run the same crowd-sourced evaluation carried out for AutoVC and show **statistically significant improvements** on both the overall conversion, that achieves a mean MOS of 3.60 ± 1.18, and for each dialect. The best results were obtained when converting to the Scottish and from the Northern accent, with MOS means of 4.02 and 3.70, respectively. We can clearly see the pattern that starting from the **Northern accent** results in better conversions for both AutoVC and

YourTTS, and we attribute this result to the fact that the Northern accent is one of the most represented in the dataset.

Objective metrics vs perceptual quality Next, we'd like to stress the little correlation we observed between **objective metrics and perceptual quality**. In particular, most of the objective metrics we reported for speaker similarity and intelligibility were not as high as expected, even though speech sounded close to the selected speaker and definitely comprehensible by a human being. We attribute this gap to multiple reasons, such as background noise, the acoustic model producing many more tokens than necessary (resulting in mumbling towards the end of sentences), the VC/AC models distorting linguistic content and the inherent noise introduced by the vocoder. All such errors add up and contribute to overall lower scores. Still, we believe that properly training each component for enough time would lead to much better objective metrics, in line with perceptual quality.

To conclude, we also want to point out that most of the results reported in this thesis could be improved by relying on more **computational power**: due to the limited resources at our disposal, we were only able to carry out relatively small scale experiments, even though most of the reported models are resource-hungry and would benefit a lot from either some kind of distributed training procedure or simply single GPUs with more capacity.

CONCLUSIONS

After extensive literature review and the introduction of modern components and tasks carried out in a full TTS scenario, it is clear that most of the approaches presented so far in the flexible TTS field focus on controlling synthesis for specific attributes, such as speaker identity and language. Yet, none of the previous work explored the expansion of TTS systems to model regional variations of a language, i.e. dialect accents.

To accomplish this objective, this work first identifies a suitable open-source dataset, namely, SLR83 [42], which is a crowd-sourced high-quality speech corpus containing over 30 h of speech in 6 different dialects of the British Isles. Then, 2 TTS pipelines are proposed. The first one is a multi-dialect extension of the popular Tacotron 2 [152] architecture, while the second one relies on a multi-speaker Tacotron 2 model followed by an AC system, which is the dialect accent adaptation of the well-established AutoVC [142] model. Both the acoustic model in the first pipeline and the AC model in the second pipeline are conditioned on TitaNet [93] dialect embeddings to control for regional accent in the synthesised speech. Moreover, such approaches are compared to the output of a multi-lingual model fine-tuned for the multi-dialect case, namely YourTTS [20].

Results show that individual components can deliver on their target task, but the combination of them in a full TTS pipeline does not yield the expected results. In particular, we show that AutoVC is suitable for the task of regional accent conversion, if properly trained and tuned for the correct bottleneck size. Moreover, the comparison of crowd-sourced evaluations, ranking systems in terms of how close synthesised speech is to the selected accent, reveals a statistically significant preference for the output produced by

YourTTS with respect to that of AutoVC, and informal listening strongly backs this conclusion.

Even though the experimental results didn't meet our expectations regarding the end-to-end flow in our proposed pipelines, the great results obtained with their constituents strongly makes us believe that more careful training strategies, such as the ones adopted with YourTTS, along with better computational resources and training times, would lead to more positive outcomes.

To further improve each component, the following research directions could be worth exploring. For dialect accent embeddings, we believe that training on large-scale corpora would help and, since open-source datasets lack in this domain, fine-tuning language encoders on dialect discrimination could be beneficial. Moreover, focusing on the disentanglement between all speech attributes we aim to control for is key, and approaches relying on adversarial training and domain adaptation, such as the ones presented in [186, 192], seem promising. For the acoustic model, we are convinced that experimentation times could be reduced by a large margin by switching from an attention-based model to one of the latest TTS architectures with external duration predictors, such as FastSpeech [145], or to smaller, more light-weight models, such as PortaSpeech [144] or neural HMMs [122]. Finally, for the AC module, it might be worth trying to experiment with more current models such as VoiceMixer [104] and StarGANv2-VC [107]. Also, given the great results obtained with YourTTS, we would like to set future directions towards the use of end-to-end models, such EATS [47] and VITS [83].

Last, we also want to highlight that the work we have done for this thesis opens up many new research tracks. Out of them all, an interesting idea would be to adapt the voice interpolation strategies adopted so far [95] to come up with novel regional accents, by mixing and matching embeddings in dialect accent latent spaces. Just imagine someone who has lived half their life in Rome and the other half in Florence: what could their accent be?

BIBLIOGRAPHY

[1] *ABI-2: The Second Accents of the British Isles Speech Corpus*. http://www.thespeechark.com/abi-2-page.html.

[2] K. Akuzawa, Y. Iwasawa, and Y. Matsuo. *Expressive Speech Synthesis via Modeling Expressions with Variational Autoencoder*. 2019. arXiv: 1804.02135 [cs.CL].

[3] O. Angelini, A. Moinet, K. Yanagisawa, and T. Drugman. *Singing Synthesis: with a little help from my attention*. 2020. arXiv: 1912.05881 [eess.AS].

[4] R. Ardila, M. Branson, K. Davis, M. Henretty, M. Kohler, J. Meyer, R. Morais, L. Saunders, F. M. Tyers, and G. Weber. *Common Voice: A Massively-Multilingual Speech Corpus*. 2020. arXiv: 1912.06670 [cs.CL].

[5] S. O. Arik et al. "Deep Voice: Real-Time Neural Text-to-Speech". In: *Proceedings of the 34th International Conference on Machine Learning - Volume 70*. ICML'17. Sydney, NSW, Australia: JMLR.org, 2017, pp. 195204.

[6] S. Ö. Arik, J. Chen, K. Peng, W. Ping, and Y. Zhou. "Neural Voice Cloning with a Few Samples". In: *CoRR* abs/1802.06006 (2018). arXiv: 1802.06006. URL: http://arxiv.org/abs/1802.06006.

[7] S. Ö. Arik, G. F. Diamos, A. Gibiansky, J. Miller, K. Peng, W. Ping, J. Raiman, and Y. Zhou. "Deep Voice 2: Multi-Speaker Neural Text-to-Speech". In: *CoRR* abs/1705.08947 (2017). arXiv: 1705.08947. URL: http://arxiv.org/abs/1705.08947.

[8] D. Bahdanau, K. Cho, and Y. Bengio. “Neural machine translation by jointly learning to align and translate”. In: *arXiv preprint arXiv:1409.0473* (2014). URL: https://arxiv.org/abs/1409.0473.

[9] Y. Bengio, P. Simard, and P. Frasconi. “Learning long-term dependencies with gradient descent is difficult”. In: *IEEE Transactions on Neural Networks* 5.2 (1994), pp. 157–166. DOI: 10.1109/72.279181.

[10] M. Bernard and H. Titeux. “Phonemizer: Text to Phones Transcription for Multiple Languages in Python”. In: *Journal of Open Source Software* 6.68 (2021), p. 3958. DOI: 10.21105/joss.03958. URL: https://doi.org/10.21105/joss.03958.

[11] L. Biewald. *Experiment Tracking with Weights and Biases*. Software available from wandb.com. 2020. URL: https://www.wandb.com/.

[12] M. Binkowski, J. Donahue, S. Dieleman, A. Clark, E. Elsen, N. Casagrande, L. C. Cobo, and K. Simonyan. “High Fidelity Speech Synthesis with Adversarial Networks”. In: *CoRR* abs/1909.11646 (2019). arXiv: 1909.11646. URL: http://arxiv.org/abs/1909.11646.

[13] D. Bourgin. *numpy-ml*. 2021. URL: https://numpy-ml.readthedocs.io/en/latest/ (visited on 09/28/2021).

[14] M. M. Bronstein, J. Bruna, T. Cohen, and P. Velickovic. “Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges”. In: *CoRR* abs/2104.13478 (2021). arXiv: 2104.13478. URL: https://arxiv.org/abs/2104.13478.

[15] R. Carlson. “Models of Speech Synthesis”. In: *Proceedings of the National Academy of Sciences* 92 (Mar. 2002). DOI: 10.1073/pnas.92.22.9932.

[16] L. Carroll. *Alices Adventures in Wonderland*. Broadview Press", 2011.

[17] R. Caruana. “Multitask Learning”. In: *Machine Learning* 28 (July 1997). DOI: 10.1023/A:1007379606734.

[18] E. Casanova, A. C. Junior, C. Shulby, F. S. d. Oliveira, J. P. Teixeira, M. A. Ponti, and S. Aluísio. “TTS-Portuguese Corpus: a corpus for speech synthesis in Brazilian Portuguese”. In: *Language Resources and Evaluation* (2022). ISSN:

1574-0218. DOI: 10.1007/s10579-021-09570-4. URL: http://dx.doi.org/10.1007/s10579-021-09570-4.

[19] E. Casanova, C. Shulby, E. Gölge, N. M. Müller, F. S. de Oliveira, A. C. Junior, A. da Silva Soares, S. M. Aluisio, and M. A. Ponti. *SC-GlowTTS: an Efficient Zero-Shot Multi-Speaker Text-To-Speech Model*. 2021. arXiv: 2104.05557 [eess.AS].

[20] E. Casanova, J. Weber, C. Shulby, A. C. Júnior, E. Gölge, and M. A. Ponti. "YourTTS: Towards Zero-Shot Multi-Speaker TTS and Zero-Shot Voice Conversion for everyone". In: *CoRR* abs/2112.02418 (2021). arXiv: 2112.02418. URL: https://arxiv.org/abs/2112.02418.

[21] M. Cerak, M. Rusko, and M. Trnka. "Diagnostic evaluation of synthetic speech using speech recognition". In: *16th International Congress on Sound and Vibration 2009, ICSV 2009* 8 (Jan. 2009), pp. 5–9.

[22] M. Chen, X. Tan, B. Li, Y. Liu, T. Qin, S. Zhao, and T.-Y. Liu. *AdaSpeech: Adaptive Text to Speech for Custom Voice*. 2021. arXiv: 2103.00993 [eess.AS].

[23] Y. Chen et al. "Sample Efficient Adaptive Text-to-Speech". In: *CoRR* abs/1809.10460 (2018). arXiv: 1809.10460. URL: http://arxiv.org/abs/1809.10460.

[24] K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio. "On the Properties of Neural Machine Translation: Encoder-Decoder Approaches". In: *CoRR* abs/1409.1259 (2014). arXiv: 1409.1259. URL: http://arxiv.org/abs/1409.1259.

[25] Y.-P. Cho, F.-R. Yang, Y.-C. Chang, C.-T. Cheng, X.-H. Wang, and Y.-W. Liu. *A Survey on Recent Deep Learning-driven Singing Voice Synthesis Systems*. 2021. arXiv: 2110.02511 [eess.AS].

[26] J. Chorowski, D. Bahdanau, D. Serdyuk, K. Cho, and Y. Bengio. "Attention-Based Models for Speech Recognition". In: *CoRR* abs/1506.07503 (2015). arXiv: 1506.07503. URL: http://arxiv.org/abs/1506.07503.

[27] O. K. D. M. K. W. Christopher Cieri David Graff. *Fisher English Training Speech Part 1 Transcripts*. 2004. DOI: 10.35111/w4bk-9b14.

[28] J. S. Chung, A. Nagrani, and A. Zisserman. "VoxCeleb2: Deep Speaker Recognition". In: *INTERSPEECH*. 2018.

[29] J. S. Chung, J. Huh, S. Mun, M. Lee, H.-S. Heo, S. Choe, C. Ham, S. Jung, B.-J. Lee, and I. Han. "In Defence of Metric Learning for Speaker Recognition". In: *Interspeech 2020* (2020). DOI: 10.21437/interspeech.2020-1064. URL: http://dx.doi.org/10.21437/Interspeech.2020-1064.

[30] T. N. community. *NumPy*. 2021. URL: https://numpy.org/doc/stable/ (visited on 09/28/2021).

[31] T. S. community. *SciPy*. 2021. URL: https://docs.scipy.org/doc/scipy/index.html (visited on 09/28/2021).

[32] E. Cooper, C.-I. Lai, Y. Yasuda, F. Fang, X. Wang, N. Chen, and J. Yamagishi. *Zero-Shot Multi-Speaker Text-To-Speech with State-of-the-art Neural Speaker Embeddings*. 2020. arXiv: 1910.10838 [eess.AS].

[33] *Corpora e Lessici dell'Italiano Parlato e Scritto (CLIPS)*. http://www.clips.unina.it/it/.

[34] M. Cotescu, T. Drugman, G. Huybrechts, J. Lorenzo-Trueba, and A. Moinet. "Voice Conversion for Whispered Speech Synthesis". In: *CoRR* abs/1912.05289 (2019). arXiv: 1912.05289. URL: http://arxiv.org/abs/1912.05289.

[35] L. J. Cronbach. "Coefficient alpha and the internal structure of tests". In: *Psychometrika* 16 (1951), pp. 297–334.

[36] K. Crowston. "Amazon Mechanical Turk: A Research Tool for Organizations and Information Systems Scholars". In: *Shaping the Future of ICT Research. Methods and Approaches*. Ed. by A. Bhattacherjee and B. Fitzgerald. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 210–221. ISBN: 978-3-642-35142-6.

[37] G. Cybenko. "Approximation by superpositions of a sigmoidal function". In: *Mathematics of Control, Signals, and Systems (MCSS)* 2.4 (Dec. 1989), pp. 303–314. ISSN: 0932-4194. DOI: 10.1007/BF02551274. URL: http://dx.doi.org/10.1007/BF02551274.

[38] A. R. Damasio. *Descartes' Error. Emotion, reason and the human brain.* 1994.

[39] A. Defossez, G. Synnaeve, and Y. Adi. *Real Time Speech Enhancement in the Waveform Domain*. 2020. arXiv: 2006.12847 [eess.AS].

[40] M. P. Deisenroth, A. A. Faisal, and C. S. Ong. *Mathematics for Machine Learning*. Cambridge University Press, 2020. URL: https://mml-book.github.io/book/mml-book.pdf.

[41] A. Demarco and S. Cox. "Iterative Classification Of Regional British Accents In I-Vector Space". English. In: Symposium on machine learning in speech and language processing ; Conference date: 01-09-2012 Through 30-09-2012. Sept. 2012, pp. 1–4.

[42] I. Demirsahin, O. Kjartansson, A. Gutkin, and C. Rivera. "Open-source Multi-speaker Corpora of the English Accents in the British Isles". In: *Proceedings of The 12th Language Resources and Evaluation Conference (LREC)*. Marseille, France: European Language Resources Association (ELRA), May 2020, pp. 6532–6541. ISBN: 979-10-95546-34-4. URL: https://www.aclweb.org/anthology/2020.lrec-1.804.

[43] J. Deng, J. Guo, N. Xue, and S. Zafeiriou. *ArcFace: Additive Angular Margin Loss for Deep Face Recognition*. 2019. arXiv: 1801.07698 [cs.CV].

[44] B. Desplanques, J. Thienpondt, and K. Demuynck. "ECAPA-TDNN: Emphasized Channel Attention, Propagation and Aggregation in TDNN Based Speaker Verification". In: *Interspeech 2020*. Ed. by H. Meng, B. Xu, and T. F. Zheng. ISCA, 2020, pp. 3830–3834.

[45] T. DeVries and G. W. Taylor. *Improved Regularization of Convolutional Neural Networks with Cutout*. 2017. arXiv: 1708.04552 [cs.CV].

[46] S. Ding, G. Zhao, and R. Gutierrez-Osuna. "Accentron: Foreign accent conversion to arbitrary non-native speakers using zero-shot learning". In: *Computer Speech Language* 72 (2022), p. 101302. ISSN: 0885-2308. DOI: https://doi.org/10.1016/j.csl.2021.101302. URL: https://www.sciencedirect.com/science/article/pii/S0885230821001029.

[47] J. Donahue, S. Dieleman, M. Binkowski, E. Elsen, and K. Simonyan. "End-to-End Adversarial Text-to-Speech". In: *CoRR* abs/2006.03575 (2020). arXiv: 2006.03575. URL: https://arxiv.org/abs/2006.03575.

[48] R. E. Donovan, M. Franz, J. S. Sorensen, and S. Roukos. "Phrase splicing and variable substitution using the IBM trainable speech synthesis system". In: *Proceedings of the 1999 IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP '99, Phoenix, Arizona, USA, March 15-19, 1999*. IEEE Computer Society, 1999, pp. 373–376. DOI: 10.1109/ICASSP.1999.758140. URL: https://doi.org/10.1109/ICASSP.1999.758140.

[49] D. Dua and C. Graff. *UCI Machine Learning Repository*. 2017. URL: http://archive.ics.uci.edu/ml.

[50] V. Dumoulin and F. Visin. "A guide to convolution arithmetic for deep learning". In: *ArXiv e-prints* (2016). eprint: 1603.07285.

[51] J. L. Elman. "Finding structure in time". In: *Cognitive Science* 14.2 (1990), pp. 179–211. ISSN: 0364-0213. DOI: https://doi.org/10.1016/0364-0213(90)90002-E. URL: https://www.sciencedirect.com/science/article/pii/036402139090002E.

[52] W. e. a. Falcon. *PyTorch Lightning*. https://github.com/PytorchLightning/pytorch-lightning. 2019. DOI: 10.5281/zenodo.3828935.

[53] D. Felps, H. Bortfeld, and R. Gutierrez-Osuna. "Foreign Accent Conversion in Computer Assisted Pronunciation Training". In: *Speech Commun.* 51.10 (2009), pp. 920932. ISSN: 0167-6393. DOI: 10.1016/j.specom.2008.11.004. URL: https://doi.org/10.1016/j.specom.2008.11.004.

[54] J. Fong, J. Wu, P. Agrawal, A. Gibiansky, T. Koehler, and Q. He. "Improving Polyglot Speech Synthesis through Multi-task and Adversarial Learning". In: *Proc. 11th ISCA Speech Synthesis Workshop (SSW 11)*. 2021, pp. 172–176. DOI: 10.21437/SSW.2021-30.

[55] *Freiburg Corpus of English Dialects (FRED)*. https://www2.anglistik.uni-freiburg.de/institut/lskortmann/FRED/.

[56] S. Furui. "Speech and Speaker Recognition Evaluation". In: Apr. 2007, pp. 1–27. ISBN: 978-1-4020-5816-5. DOI: 10.1007/978-1-4020-5817-2_1.

[57] M. A. I. L. GmbH. *The M-AILABS speech dataset*. https://www.caito.de/2019/01/the-m-ailabs-speech-dataset/. 2017.

[58] J. J. Godfrey and E. Holliman. *Switchboard-1 Release 2*. 1993. DOI: 10.35111/sw3h-rw02.

[59] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[60] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. *Generative Adversarial Networks*. 2014. arXiv: 1406.2661 [stat.ML].

[61] Google. *Colaboratory*. URL: \url{https://colab.research.google.com/}.

[62] D. Griffin and J. Lim. "Signal estimation from modified short-time Fourier transform". In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 32.2 (1984), pp. 236–243. DOI: 10.1109/TASSP.1984.1164317.

[63] W. Han, Z. Zhang, Y. Zhang, J. Yu, C.-C. Chiu, J. Qin, A. Gulati, R. Pang, and Y. Wu. *ContextNet: Improving Convolutional Neural Networks for Automatic Speech Recognition with Global Context*. 2020. arXiv: 2005.03191 [eess.AS].

[64] M. He, J. Yang, L. He, and F. K. Soong. "Multilingual Byte2Speech Models for Scalable Low-resource Speech Synthesis". In: *CoRR* abs/2103.03541 (2021). arXiv: 2103.03541. URL: https://arxiv.org/abs/2103.03541.

[65] G. Heigold, I. Moreno, S. Bengio, and N. Shazeer. *End-to-End Text-Dependent Speaker Verification*. 2015. arXiv: 1509.08062 [cs.LG].

[66] H. Hemati and D. Borth. "Using IPA-Based Tacotron for Data Efficient Cross-Lingual Speaker Adaptation and Pronunciation Enhancement". In: *CoRR* abs/2011.06392 (2020). arXiv: 2011.06392. URL: https://arxiv.org/abs/2011.06392.

[67] H. S. Heo, B.-J. Lee, J. Huh, and J. S. Chung. *Clova Baseline System for the VoxCeleb Speaker Recognition Challenge 2020*. 2020. arXiv: 2009.14153 [eess.AS].

[68] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, G. Klambauer, and S. Hochreiter. "GANs Trained by a Two Time-Scale Update Rule Converge to a Nash Equilibrium". In: *CoRR* abs/1706.08500 (2017). arXiv: 1706.08500. URL: http://arxiv.org/abs/1706.08500.

[69] S. Hochreiter and J. Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[70] J. Hu, L. Shen, and G. Sun. "Squeeze-and-Excitation Networks". In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2018, pp. 7132–7141. DOI: 10.1109/CVPR.2018.00745.

[71] W. Huang, E. Cooper, J. Yamagishi, and T. Toda. "LDNet: Unified Listener Dependent Modeling in MOS Prediction for Synthetic Speech". In: *CoRR* abs/2110.09103 (2021). arXiv: 2110.09103. URL: https://arxiv.org/abs/2110.09103.

[72] A. J. Hunt and A. W. Black. "Unit selection in a concatenative speech synthesis system using a large speech database". In: *1996 IEEE International Conference on Acoustics, Speech, and Signal Processing Conference Proceedings* 1 (1996), 373–376 vol. 1.

[73] J. D. Hunter. "Matplotlib: A 2D graphics environment". In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55.

[74] S. Ioffe and C. Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. arXiv: 1502.03167 [cs.LG].

[75] K. Ito and L. Johnson. *The LJ Speech Dataset*. https://keithito.com/LJ-Speech-Dataset/. 2017.

[76] M. A. Jette, A. B. Yoo, and M. Grondona. "SLURM: Simple Linux Utility for Resource Management". In: *In Lecture Notes in Computer Science: Proceedings of Job Scheduling Strategies for Parallel Processing (JSSPP) 2003*. Springer-Verlag, 2002, pp. 44–60.

[77] Y. Jia et al. "Transfer Learning from Speaker Verification to Multispeaker Text-To-Speech Synthesis". In: *CoRR* abs/1806.04558 (2018). arXiv: 1806.04558. URL: http://arxiv.org/abs/1806.04558.

[78] B. Joshi, A. Chetan, P. Madaan, P. Jain, S. Anand, Eshita, and S. Singh. *An exploration into Deep Learning methods for Emotional Text-to-Speech*. Version v1.0.0. June 2020. DOI: 10.5281/zenodo.3876081. URL: https://doi.org/10.5281/zenodo.3876081.

[79] D. Jurafsky and J. H. Martin. *Speech and Language Processing (3rd Edition Draft)*. 2021. URL: https://web.stanford.edu/~jurafsky/slp3/.

[80] H. Kawahara. "STRAIGHT, exploitation of the other aspect of VOCODER: Perceptually isomorphic decomposition of speech sounds". In: *Acoustical Science and Technology* 27.6 (2006), pp. 349–353. DOI: 10.1250/ast.27.349.

[81] K. Kilgour, M. Zuluaga, D. Roblek, and M. Sharifi. *Fréchet Audio Distance: A Metric for Evaluating Music Enhancement Algorithms*. 2019. arXiv: 1812.08466 [eess.AS].

[82] J. Kim, S. Kim, J. Kong, and S. Yoon. *Glow-TTS: A Generative Flow for Text-to-Speech via Monotonic Alignment Search*. 2020. arXiv: 2005.11129 [eess.AS].

[83] J. Kim, J. Kong, and J. Son. "Conditional Variational Autoencoder with Adversarial Learning for End-to-End Text-to-Speech". In: *CoRR* abs/2106.06103 (2021). arXiv: 2106.06103. URL: https://arxiv.org/abs/2106.06103.

[84] T.-H. Kim, S. Cho, S. Choi, S. Park, and S.-Y. Lee. *Emotional Voice Conversion using Multitask Learning with Text-to-speech*. 2019. arXiv: 1911.06149 [eess.AS].

[85] D. P. Kingma and J. Ba. "Adam: A Method for Stochastic Optimization". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Y. Bengio and Y. LeCun. 2015. URL: http://arxiv.org/abs/1412.6980.

[86] D. P. Kingma, D. J. Rezende, S. Mohamed, and M. Welling. "Semi-Supervised Learning with Deep Generative Models". In: *CoRR* abs/1406.5298 (2014). arXiv: 1406.5298. URL: http://arxiv.org/abs/1406.5298.

[87] D. P. Kingma and M. Welling. *Auto-Encoding Variational Bayes*. 2014. arXiv: 1312.6114 [stat.ML].

[88] D. H. Klatt. "Review of text-to-speech conversion for English". In: *The Journal of the Acoustical Society of America* 82.3 (1987), pp. 737–793.

[89] T. Kluyver et al. "Jupyter Notebooks – a publishing format for reproducible computational workflows". In: *Positioning and Power in Academic Publishing: Players, Agents and Agendas*. Ed. by F. Loizides and B. Schmidt. IOS Press. 2016, pp. 87 –90.

[90] T. Ko, V. Peddinti, D. Povey, M. L. Seltzer, and S. Khudanpur. "A study on data augmentation of reverberant speech for robust speech recognition". In: *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2017, pp. 5220–5224. DOI: 10.1109/ICASSP.2017.7953152.

[91] I. Kobyzev, S. J. Prince, and M. A. Brubaker. "Normalizing Flows: An Introduction and Review of Current Methods". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 43.11 (2021), pp. 39643979. ISSN: 1939-3539. DOI: 10.1109/tpami.2020.2992934. URL: http://dx.doi.org/10.1109/TPAMI.2020.2992934.

[92] N. R. Koluguri, J. Li, V. Lavrukhin, and B. Ginsburg. *SpeakerNet: 1D Depth-wise Separable Convolutional Network for Text-Independent Speaker Recognition and Verification*. 2020. arXiv: 2010.12653 [eess.AS].

[93] N. R. Koluguri, T. Park, and B. Ginsburg. *TitaNet: Neural Model for speaker representation with 1D Depth-wise separable convolutions and global context*. 2021. arXiv: 2110.04410 [eess.AS].

[94] J. Kong, J. Kim, and J. Bae. *HiFi-GAN: Generative Adversarial Networks for Efficient and High Fidelity Speech Synthesis*. 2020. arXiv: 2010.05646 [cs.SD].

[95] R. Korostik, J. Latorre, S. Achanta, and Y. Stylianou. "Assessing Speaker Interpolation in Neural Text-to-Speech". In: *Speech and Computer*. Ed. by A. Karpov and R. Potapova. Cham: Springer International Publishing, 2021, pp. 360–371. ISBN: 978-3-030-87802-3.

[96] B. Kortmann and E. W. Schneider, eds. *A Handbook of Varieties of English: A Multimedia Reference Tool. Volume 1: Phonology. Volume 2: Morphology and Syntax*. De Gruyter Mouton, 2008. ISBN: 9783110197181. DOI:

doi:10.1515/9783110197181. URL: https://doi.org/10.1515/9783110197181.

[97] S. Kriman, S. Beliaev, B. Ginsburg, J. Huang, O. Kuchaiev, V. Lavrukhin, R. Leary, J. Li, and Y. Zhang. *QuartzNet: Deep Automatic Speech Recognition with 1D Time-Channel Separable Convolutions*. 2019. arXiv: 1910.10261 [eess.AS].

[98] R. Kubichek. "Mel-cepstral distance measure for objective speech quality assessment". In: *Proceedings of IEEE Pacific Rim Conference on Communications Computers and Signal Processing*. Vol. 1. 1993, 125–128 vol.1. DOI: 10.1109/PACRIM.1993.407206.

[99] O. Kuchaiev et al. "NeMo: a toolkit for building AI applications using Neural Modules". In: *CoRR* abs/1909.09577 (2019). arXiv: 1909.09577. URL: http://arxiv.org/abs/1909.09577.

[100] K. Kumar, R. Kumar, T. de Boissiere, L. Gestin, W. Z. Teoh, J. Sotelo, A. de Brebisson, Y. Bengio, and A. Courville. *MelGAN: Generative Adversarial Networks for Conditional Waveform Synthesis*. 2019. arXiv: 1910.06711 [eess.AS].

[101] A. Lamb, A. Goyal, Y. Zhang, S. Zhang, A. Courville, and Y. Bengio. *Professor Forcing: A New Algorithm for Training Recurrent Networks*. 2016. arXiv: 1610.09038 [stat.ML].

[102] A. B. L. Larsen, S. K. Sønderby, H. Larochelle, and O. Winther. *Autoencoding beyond pixels using a learned similarity metric*. 2016. arXiv: 1512.09300 [cs.LG].

[103] J. Latorre, J. Lachowicz, J. Lorenzo-Trueba, T. Merritt, T. Drugman, S. Ronanki, and K. Viacheslav. "Effect of data reduction on sequence-to-sequence neural TTS". In: *CoRR* abs/1811.06315 (2018). arXiv: 1811.06315. URL: http://arxiv.org/abs/1811.06315.

[104] S.-H. Lee, J.-H. Kim, H. Chung, and S.-W. Lee. "VoiceMixer: Adversarial Voice Style Mixup". In: *Advances in Neural Information Processing Systems*. Ed. by A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan. 2021. URL: https://openreview.net/forum?id=0lzmTb4LGd3F.

[105] Y. Leng, X. Tan, S. Zhao, F. K. Soong, X. Li, and T. Qin. "MBNet: MOS Prediction for Synthesized Speech with Mean-Bias Network". In: *CoRR* abs/2103.00110 (2021). arXiv: 2103.00110. URL: https://arxiv.org/abs/2103.00110.

[106] T. Li, S. Yang, L. Xue, and L. Xie. *Controllable Emotion Transfer For End-to-End Speech Synthesis*. 2020. arXiv: 2011.08679 [cs.SD].

[107] Y. A. Li, A. Zare, and N. Mesgarani. "StarGANv2-VC: A Diverse, Unsupervised, Non-parallel Framework for Natural-Sounding Voice Conversion". In: *CoRR* abs/2107.10394 (2021). arXiv: 2107.10394. URL: https://arxiv.org/abs/2107.10394.

[108] *LibriVox*. https://librivox.org/.

[109] J. H. Lim and J. C. Ye. *Geometric GAN*. 2017. arXiv: 1705.02894 [stat.ML].

[110] M. Lin, Q. Chen, and S. Yan. *Network In Network*. 2014. arXiv: 1312.4400 [cs.NE].

[111] P. Liu, X. Wu, S. Kang, G. Li, D. Su, and D. Yu. *Maximizing Mutual Information for Tacotron*. 2019. arXiv: 1909.01145 [eess.AS].

[112] C. Lo, S. Fu, W. Huang, X. Wang, J. Yamagishi, Y. Tsao, and H. Wang. "MOSNet: Deep Learning based Objective Assessment for Voice Conversion". In: *CoRR* abs/1904.08352 (2019). arXiv: 1904.08352. URL: http://arxiv.org/abs/1904.08352.

[113] P. C. Loizou. "Speech Quality Assessment". In: *Multimedia Analysis, Processing and Communications*. Ed. by W. Lin, D. Tao, J. Kacprzyk, Z. Li, E. Izquierdo, and H. Wang. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 623–654. ISBN: 978-3-642-19551-8. DOI: 10.1007/978-3-642-19551-8_23. URL: https://doi.org/10.1007/978-3-642-19551-8_23.

[114] J. Lorenzo-Trueba, T. Drugman, J. Latorre, T. Merritt, B. Putrycz, R. Barra-Chicote, A. Moinet, and V. Aggarwal. *Towards achieving robust universal neural vocoding*. 2019. arXiv: 1811.06292 [eess.AS].

[115] I. Loshchilov and F. Hutter. "Fixing Weight Decay Regularization in Adam". In: *CoRR* abs/1711.05101 (2017). arXiv: 1711.05101. URL: http://arxiv.org/abs/1711.05101.

[116] R. Luo, X. Tan, R. Wang, T. Qin, J. Li, S. Zhao, E. Chen, and T. Liu. "LightSpeech: Lightweight and Fast Text to Speech with Neural Architecture Search". In: *CoRR* abs/2102.04040 (2021). arXiv: 2102.04040. URL: https://arxiv.org/abs/2102.04040.

[117] L. van der Maaten and G. Hinton. "Visualizing Data using t-SNE". In: *Journal of Machine Learning Research* 9.86 (2008), pp. 2579–2605. URL: http://jmlr.org/papers/v9/vandermaaten08a.html.

[118] Y. Masuyama, K. Yatabe, Y. Koizumi, Y. Oikawa, and N. Harada. "Deep Griffin-Lim Iteration". In: *CoRR* abs/1903.03971 (2019). arXiv: 1903.03971. URL: http://arxiv.org/abs/1903.03971.

[119] B. McFee, C. Raffel, D. Liang, D. P. Ellis, M. McVicar, E. Battenberg, and O. Nieto. "librosa: Audio and music signal analysis in python". In: *Proceedings of the 14th python in science conference*. Vol. 8. 2015.

[120] L. McInnes, J. Healy, and J. Melville. *UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction*. 2020. arXiv: 1802.03426 [stat.ML].

[121] Wes McKinney. "Data Structures for Statistical Computing in Python". In: *Proceedings of the 9th Python in Science Conference*. Ed. by Stéfan van der Walt and Jarrod Millman. 2010, pp. 56 –61. DOI: 10.25080/Majora-92bf1922-00a.

[122] S. Mehta, É. Székely, J. Beskow, and G. E. Henter. "Neural HMMs are all you need (for high-quality attention-free TTS)". In: *ArXiv* abs/2108.13320 (2021).

[123] C. Miao, S. Liang, Z. Liu, M. Chen, J. Ma, S. Wang, and J. Xiao. *EfficientTTS: An Efficient and High-Quality Text-to-Speech Architecture*. 2020. arXiv: 2012.03500 [eess.AS].

[124] M. Morise. "D4C, a band-aperiodicity estimator for high-quality speech synthesis". In: *Speech Communication* 84 (2016), pp. 57–65. ISSN: 0167-6393. DOI: https://doi.org/10.1016/j.specom.2016.09.001. URL:

https://www.sciencedirect.com/science/article/pii/S0167639316300413.

[125] M. Morise, F. Yokomori, and K. Ozawa. "WORLD: A Vocoder-Based High-Quality Speech Synthesis System for Real-Time Applications". In: *IEICE Transactions on Information and Systems* E99.D.7 (2016), pp. 1877–1884. DOI: 10.1587/transinf.2015EDP7457.

[126] E. Nachmani and L. Wolf. "Unsupervised Polyglot Text To Speech". In: *CoRR* abs/1902.02263 (2019). arXiv: 1902.02263. URL: http://arxiv.org/abs/1902.02263.

[127] A. Nagrani, J. S. Chung, and A. Zisserman. "VoxCeleb: a large-scale speaker identification dataset". In: *INTERSPEECH*. 2017.

[128] M. Najafian, S. Safavi, P. Weber, and M. Russell. "Identification of British English regional accents using fusion of i-vector and multi-accent phonotactic systems". In: *Proc. The Speaker and Language Recognition Workshop (Odyssey 2016)*. 2016, pp. 132–139. DOI: 10.21437/Odyssey.2016-19.

[129] T. Nekvinda and O. Duek. *One Model, Many Languages: Meta-learning for Multilingual Text-to-Speech*. 2020. arXiv: 2008.00768 [eess.AS].

[130] K. Okabe, T. Koshinaka, and K. Shinoda. "Attentive Statistics Pooling for Deep Speaker Embedding". In: *Interspeech 2018* (2018). DOI: 10.21437/interspeech.2018-993. URL: http://dx.doi.org/10.21437/Interspeech.2018-993.

[131] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. W. Senior, and K. Kavukcuoglu. "WaveNet: A Generative Model for Raw Audio". In: *CoRR* abs/1609.03499 (2016). arXiv: 1609.03499. URL: http://arxiv.org/abs/1609.03499.

[132] A. van den Oord, N. Kalchbrenner, O. Vinyals, L. Espeholt, A. Graves, and K. Kavukcuoglu. "Conditional Image Generation with PixelCNN Decoders". In: *CoRR* abs/1606.05328 (2016). arXiv: 1606.05328. URL: http://arxiv.org/abs/1606.05328.

[133] D. D. Palmer and M. A. Hearst. "Adaptive Multilingual Sentence Boundary Disambiguation". In: *Computational Linguistics* 23.2 (1997), pp. 241–267. URL: https://aclanthology.org/J97-2002.

[134] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur. "Librispeech: An ASR corpus based on public domain audio books". In: *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2015, pp. 5206–5210. DOI: 10.1109/ICASSP.2015.7178964.

[135] D. S. Park, W. Chan, Y. Zhang, C.-C. Chiu, B. Zoph, E. D. Cubuk, and Q. V. Le. "SpecAugment: A Simple Data Augmentation Method for Automatic Speech Recognition". In: *Interspeech 2019* (2019). DOI: 10.21437/interspeech.2019-2680. URL: http://dx.doi.org/10.21437/Interspeech.2019-2680.

[136] A. Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[137] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[138] N. Perraudin, P. Balazs, and P. L. Søndergaard. "A fast Griffin-Lim algorithm". In: *2013 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*. 2013, pp. 1–4. DOI: 10.1109/WASPAA.2013.6701851.

[139] W. Ping, K. Peng, A. Gibiansky, S. Ö. Arik, A. Kannan, S. Narang, J. Raiman, and J. Miller. "Deep Voice 3: 2000-Speaker Neural Text-to-Speech". In: *CoRR* abs/1710.07654 (2017). arXiv: 1710.07654. URL: http://arxiv.org/abs/1710.07654.

[140] M. Przybocki and A. Martin. "NIST's Assessment of Text Independent Speaker Recognition Performance". en. In: (2002).

[141] K. Qian, Y. Zhang, S. Chang, D. Cox, and M. Hasegawa-Johnson. *Unsupervised Speech Decomposition via Triple Information Bottleneck*. 2021. arXiv: 2004.11284 [eess.AS].

[142] K. Qian, Y. Zhang, S. Chang, X. Yang, and M. Hasegawa-Johnson. *AUTOVC: Zero-Shot Voice Style Transfer with Only Autoencoder Loss*. 2019. arXiv: 1905.05879 [eess.AS].

[143] M. Ravanelli et al. *SpeechBrain: A General-Purpose Speech Toolkit*. arXiv:2106.04624. 2021. arXiv: 2106.04624 [eess.AS].

[144] Y. Ren, J. Liu, and Z. Zhao. "PortaSpeech: Portable and High-Quality Generative Text-to-Speech". In: *ArXiv* abs/2109.15166 (2021).

[145] Y. Ren, Y. Ruan, X. Tan, T. Qin, S. Zhao, Z. Zhao, and T. Liu. "FastSpeech: Fast, Robust and Controllable Text to Speech". In: *CoRR* abs/1905.09263 (2019). arXiv: 1905.09263. URL: http://arxiv.org/abs/1905.09263.

[146] A. Rix, J. Beerends, M. Hollier, and A. Hekstra. "Perceptual evaluation of speech quality (PESQ)-a new method for speech quality assessment of telephone networks and codecs". In: *2001 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No.01CH37221)*. Vol. 2. 2001, 749–752 vol.2. DOI: 10.1109/ICASSP.2001.941023.

[147] O. Ronneberger, P. Fischer, and T. Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: *CoRR* abs/1505.04597 (2015). arXiv: 1505.04597. URL: http://arxiv.org/abs/1505.04597.

[148] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. 3rd ed. Prentice Hall, 2010.

[149] X. S D'Arcy and M. Russell. "Experiments with the ABI (Accents of the British Isles) Speech Corpus". English. In: Proc. Interspeech 2008 ; Conference date: 01-01-2008. Jan. 2008, pp. 293–296.

[150] F. E. Satterthwaite. "An Approximate Distribution of Estimates of Variance Components". In: *Biometrics Bulletin* 2.6 (1946), pp. 110–114. ISSN: 00994987. URL: http://www.jstor.org/stable/3002019.

[151] M. Schoeffler, S. Bartoschek, F.-R. Stöter, M. Roess, S. Westphal, B. Edler, and J. Herre. "webMUSHRA A Comprehensive Framework for Web-based Listening Tests". In: *Journal of Open Research Software* 6 (Feb. 2018). DOI: 10.5334/jors.187.

[152] J. Shen et al. "Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions". In: *CoRR* abs/1712.05884 (2017). arXiv: 1712.05884. URL: http://arxiv.org/abs/1712.05884.

[153] V. L. Shuby Deshpande Mantek Singh Chadha. *Audio style transfer for accents*. 2019. URL: https://shuby.de/files/11785_project.pdf.

[154] H. Silén, E. Helander, and M. Gabbouj. "Prediction of Voice Aperiodicity Based on Spectral Representations in HMM Speech Synthesis." In: Jan. 2011, pp. 105–108.

[155] R. J. Skerry-Ryan, E. Battenberg, Y. Xiao, Y. Wang, D. Stanton, J. Shor, R. J. Weiss, R. Clark, and R. A. Saurous. "Towards End-to-End Prosody Transfer for Expressive Speech Synthesis with Tacotron". In: *CoRR* abs/1803.09047 (2018). arXiv: 1803.09047. URL: http://arxiv.org/abs/1803.09047.

[156] S. L. Smith, E. Elsen, and S. De. *On the Generalization Benefit of Noise in Stochastic Gradient Descent*. 2020. arXiv: 2006.15081 [cs.LG].

[157] D. Snyder, D. Garcia-Romero, A. McCree, G. Sell, D. Povey, and S. Khudanpur. "Spoken Language Recognition using X-vectors". In: (2018), pp. 105–111.

[158] D. Snyder, D. Garcia-Romero, D. Povey, and S. Khudanpur. "Deep Neural Network Embeddings for Text-Independent Speaker Verification". In: *Proc. Interspeech 2017*. 2017, pp. 999–1003. DOI: 10.21437/Interspeech.2017-620.

[159] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

[160] B. Story. "History of speech synthesis". In: Jan. 2019, pp. 9–33. DOI: 10.4324/9780429056253-2.

[161] I. Sutskever, O. Vinyals, and Q. V. Le. "Sequence to Sequence Learning with Neural Networks". In: *CoRR* abs/1409.3215 (2014). arXiv: 1409.3215. URL: http://arxiv.org/abs/1409.3215.

[162] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. "Going Deeper with Convolutions". In: *CoRR* abs/1409.4842 (2014). arXiv: 1409.4842. URL: http://arxiv.org/abs/1409.4842.

[163] S. Takamichi, Y. Saito, N. Takamune, D. Kitamura, and H. Saruwatari. "Phase reconstruction from amplitude spectrograms based on von-Mises-distribution deep neural network". In: *CoRR* abs/1807.03474 (2018). arXiv: 1807.03474. URL: http://arxiv.org/abs/1807.03474.

[164] C. Tallec and Y. Ollivier. "Can recurrent neural networks warp time?" In: *CoRR* abs/1804.11188 (2018). arXiv: 1804.11188. URL: http://arxiv.org/abs/1804.11188.

[165] X. Tan, T. Qin, F. Soong, and T.-Y. Liu. *A Survey on Neural Speech Synthesis*. 2021. arXiv: 2106.15561 [eess.AS].

[166] P. Taylor. *Text-to-Speech Synthesis*. Cambridge University Press, 2009. DOI: 10.1017/CBO9780511816338.

[167] S. Team. *Silero Models: pre-trained enterprise-grade STT / TTS models and benchmarks*. https://github.com/snakers4/silero-models. 2021.

[168] S. Team. *Silero VAD: pre-trained enterprise-grade Voice Activity Detector (VAD), Number Detector and Language Classifier*. https://github.com/snakers4/silero-vad. 2021.

[169] *The Accents of the British Isles (ABI-1) Speech Corpus*. http://www.thespeechark.com/abi-1-page.html.

[170] *The IViE Corpus - English Intonation in the British Isles*. http://www.phon.ox.ac.uk/files/apps/IViE/.

[171] L. Theis, A. van den Oord, and M. Bethge. *A note on the evaluation of generative models*. 2016. arXiv: 1511.01844 [stat.ML].

[172] N. Tishby, F. C. Pereira, and W. Bialek. *The information bottleneck method*. 2000. arXiv: physics/0004057 [physics.data-an].

[173] N. Tits, K. E. Haddad, and T. Dutoit. *Laughter Synthesis: Combining Seq2seq modeling with Transfer Learning*. 2020. arXiv: 2008.09483 [eess.AS].

[174] H. Traunmüller. "Analytical expressions for the tonotopic sensory scale". In: *Journal of the Acoustical Society of America* 88 (1990), pp. 97–100.

[175] T. Tu, Y. Chen, C. Yeh, and H. Lee. "End-to-end Text-to-speech for Low-resource Languages by Cross-Lingual Transfer Learning". In: *CoRR* abs/1904.06508 (2019). arXiv: 1904.06508. URL: http://arxiv.org/abs/1904.06508.

[176] A. University. *Introduction to Speech Processing - Aalto University Wiki*. 2021. URL: https://wiki.aalto.fi/display/ITSP/Introduction+to+Speech+Processing (visited on 09/24/2021).

[177] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. "Attention Is All You Need". In: *CoRR* abs/1706.03762 (2017). arXiv: 1706.03762. URL: http://arxiv.org/abs/1706.03762.

[178] C. Veaux, J. Yamagishi, and S. King. "The voice bank corpus: Design, collection and data analysis of a large regional accent speech database". In: *2013 International Conference Oriental COCOSDA held jointly with 2013 Conference on Asian Spoken Language Research and Evaluation (O-COCOSDA/CASLRE)*. 2013, pp. 1–4. DOI: 10.1109/ICSDA.2013.6709856.

[179] *VoxForge*. http://www.voxforge.org/.

[180] P. Wagner et al. "Speech Synthesis Evaluation State-of-the-Art Assessment and Suggestion for a Novel Research Program". In: *Proc. 10th ISCA Workshop on Speech Synthesis (SSW 10)*. 2019, pp. 105–110. DOI: 10.21437/SSW.2019-19.

[181] L. Wan, Q. Wang, A. Papir, and I. L. Moreno. *Generalized End-to-End Loss for Speaker Verification*. 2020. arXiv: 1710.10467 [eess.AS].

[182] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman. "GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding". In: In the Proceedings of ICLR. 2019.

[183] C. Wang, Y. Tang, X. Ma, A. Wu, D. Okhonko, and J. Pino. "fairseq S2T: Fast Speech-to-Text Modeling with fairseq". In: *Proceedings of the 2020 Conference of the Asian Chapter of the Association for Computational Linguistics (AACL): System Demonstrations*. 2020.

[184] Y. Wang, D. Stanton, Y. Zhang, R. Skerry-Ryan, E. Battenberg, J. Shor, Y. Xiao, F. Ren, Y. Jia, and R. A. Saurous. *Style Tokens: Unsupervised Style Modeling, Control and Transfer in End-to-End Speech Synthesis*. 2018. arXiv: 1803.09017 [cs.CL].

[185] Y. Wang et al. "Tacotron: A Fully End-to-End Text-To-Speech Synthesis Model". In: *CoRR* abs/1703.10135 (2017). arXiv: 1703.10135. URL: http://arxiv.org/abs/1703.10135.

[186] Z. Wang, W. Ge, X. Wang, S. Yang, W. Gan, H. Chen, H. Li, L. Xie, and X. Li. "Accent and Speaker Disentanglement in Many-to-many Voice Conversion". In: *CoRR* abs/2011.08609 (2020). arXiv: 2011.08609. URL: https://arxiv.org/abs/2011.08609.

[187] M. L. Waskom. "seaborn: statistical data visualization". In: *Journal of Open Source Software* 6.60 (2021), p. 3021. DOI: 10.21105/joss.03021. URL: https://doi.org/10.21105/joss.03021.

[188] B. L. Welch. "The Generalization of 'Student's' Problem when Several Different Population Variances are Involved". In: *Biometrika* 34.1/2 (1947), pp. 28–35. ISSN: 00063444. URL: http://www.jstor.org/stable/2332510.

[189] I. Wikimedia Foundation. *Wikipedia*. 2021. URL: https://en.wikipedia.org/wiki (visited on 09/28/2021).

[190] R. J. Williams and D. Zipser. "A Learning Algorithm for Continually Running Fully Recurrent Neural Networks". In: *Neural Computation* 1.2 (1989), pp. 270–280. DOI: 10.1162/neco.1989.1.2.270.

[191] T. Wolf et al. "HuggingFace's Transformers: State-of-the-art Natural Language Processing". In: *CoRR* abs/1910.03771 (2019). arXiv: 1910.03771. URL: http://arxiv.org/abs/1910.03771.

[192] D. Xin, T. Komatsu, S. Takamichi, and H. Saruwatari. "Disentangled Speaker and Language Representations Using Mutual Information Minimization and Domain Adaptation for Cross-Lingual TTS". In: *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2021, pp. 6608–6612. DOI: 10.1109/ICASSP39728.2021.9414226.

[193] D. Xin, Y. Saito, S. Takamichi, T. Koriyama, and H. Saruwatari. "Cross-Lingual Speaker Adaptation Using Domain Adaptation and Speaker Consistency Loss for Text-To-Speech Synthesis". In: *Proc. Interspeech 2021*. 2021, pp. 1614–1618. DOI: 10.21437/Interspeech.2021-897.

[194] C. M. K. Yamagishi Junichi; Veaux. *CSTR VCTK Corpus: English Multi-speaker Corpus for CSTR Voice Cloning Toolkit (version 0.92)*. 2019. DOI: https://doi.org/10.7488/ds/2645.

[195] Y.-Y. Yang et al. "TorchAudio: Building Blocks for Audio and Speech Processing". In: *arXiv preprint arXiv:2110.15018* (2021).

[196] B. Yegnanarayana, C. d'Alessandro, and V. Darsinos. "An iterative algorithm for decomposition of speech signals into periodic and aperiodic components". In: *IEEE Transactions on Speech and Audio Processing* 6.1 (1998), pp. 1–11. DOI: 10.1109/89.650304.

[197] F. Yu and V. Koltun. "Multi-Scale Context Aggregation by Dilated Convolutions". In: *ICLR*. 2016.

[198] Y. Zhang, E. Bakhturina, and B. Ginsburg. "NeMo (Inverse) Text Normalization: From Development to Production". In: *Proc. Interspeech 2021*. 2021, pp. 4857–4859.

[199] Y. Zhang, E. Bakhturina, K. Gorman, and B. Ginsburg. "NeMo Inverse Text Normalization: From Development to Production". In: *Proc. Interspeech 2021*. 2021, pp. 4468–4472. DOI: 10.21437/Interspeech.2021-1571.

[200] Y. Zhang, R. J. Weiss, H. Zen, Y. Wu, Z. Chen, R. J. Skerry-Ryan, Y. Jia, A. Rosenberg, and B. Ramabhadran. "Learning to Speak Fluently in a Foreign Language: Multilingual Speech Synthesis and Cross-Language Voice Cloning". In: *CoRR* abs/1907.04448 (2019). arXiv: 1907.04448. URL: http://arxiv.org/abs/1907.04448.