Pedro Vide Simões

SPEECH SYNTHESIS FRAMEWORK

UNIVERSIDADE Ð COIMBRA

UNIVERSIDADE Ð
COIMBRA

Pedro Vide Simões

SPEECH SYNTHESIS FRAMEWORK

Internship report in the context of the Master in Informatics Engineering, Specialization in Intelligent Systems advised by Prof. Amílcar Cardoso from the Department of Informatics Engineering, co-advised by Prof. Fernando Perdigão from the Department of Electrical and Computers Engineering and Pedro Verruma and Bruno Antunes from Talkdesk and presented to the Faculty of Sciences and Technology / Department of Informatics Engineering.

June 2020

Faculty of Sciences and Technology
Department of Informatics Engineering

Speech Synthesis Framework

Pedro Vide Simões

Internship report in the context of the Master in Informatics Engineering, Specialization in Intelligent Systems advised by Prof. Amílcar Cardoso from the Department of Informatics Engineering, co-advised by Prof. Fernando Perdigão from the Department of Electrical and Computers Engineering and Pedro Verruma and Bruno Antunes from Talkdesk and presented to the
Faculty of Sciences and Technology / Department of Informatics Engineering.

June 2020

UNIVERSIDADE Ð
COIMBRA

This page is intentionally left blank.

Acknowledgements

Sendo este trabalho a conclusão de mais uma etapa, gostaria de deixar alguns agradecimentos.

Quero começar por agradecer ao Bruno Antunes, Pedro Verruma, Liliana Medina e Marcos Pires não só por todo apoio, disponibilidade, ensinamentos e conselhos mas também pela forma como me receberam na Talkdesk.

Aos Professores Amílcar Cardoso e Fernando Perdigão, por me terem acompanhado constantemente durante todo este trabalho, estando sempre disponíveis e prontos para ajudar, esclarecer e motivar.

A toda a minha família por todo o suporte, por serem o meu pilar e acima de tudo pelo apoio em cada decisão que eu tomo, não só em relação a este projeto mas em toda a minha vida. Obrigado à Maria, Rafael, Gabriel e Enzo por me fazerem querer alcançar sempre mais. Especial agradecimento aos meus pais, e avó, por serem quem são e nunca terem falhado para comigo. São, e sempre foram, a minha maior fonte de inspiração e motivação.

Quero agradecer à Beatriz Precatado por estar lá sempre para mim. Por me ouvir, nos bons e maus momentos, e pelo apoio constante. Por não me deixar duvidar de mim próprio e me colocar sempre um sorriso na cara. Digo sinceramente que sem ti, não teria conseguido.

Agradeço igualmente à família Precatado da Silva, por se terem tornado uma segunda familia para mim e por me ajudarem sempre que necessário.

A todos os meus amigos, com especial carinho ao Renato, Luís e Guilherme, pelo apoio, motivação e todas as palavras amigas que foram ditas quando mais precisei.

Aos restantes membros da equipa da Talkdesk por todos os momentos quer dentro, quer fora do trabalho.

This page is intentionally left blank.

Abstract

The development of systems capable of understanding and synthesizing speech has seen great progress in the last years, to a degree where such systems, more commonly known as virtual assistants, are present in most of smartphones and computers used today. These assistants are a conjunction between speech-to-text and text-to-speech systems, which allow interaction using natural language, providing an easier and more intuitive way to communicate between the user and the machine.

Seeing these quick technological advances, Talkdesk created the Virtual Agent project, aiming at the development of a system capable of answering simple and recurrent questions on call-centers, allowing human agents to deal with more complex matters and, as a consequence, optimize time and resources.

While the initial goal was the development of an in-house solution, all the research made during the first internship allowed us to gather technical knowledge on the text-to-speech field, bringing the conclusion that to make improvements over the existing open-source implementations, a considerable amount of financial and temporal resources would be needed. As such, a new, useful approach, was devised.

This dissertation proposes the development of an in-house deep learning speech synthesis framework, for Virtual Agent's text-to-speech module, aimed at detecting synthesization errors and evaluate given text-to-speech solutions. The final goal is gathering knowledge on how a certain system performs when synthesizing speech, looking at usual errors such as robotic tones, presence of extensive silence mid phrases, among others.

Keywords

Virtual Agents, Text-to-Speech, Neural Networks, Deep Learning, Natural Language Processing (NLP), Speech Synthesis, Spectrograms, Error Detection, System Evaluation

This page is intentionally left blank.

Resumo

O desenvolvimento de sistemas capazes de compreender e sintetizar fala tem visto grandes progressos nos últimos anos, sendo que estes sistemas, mais conhecidos por assistentes virtuais, já estão presentes em grande parte dos telemóveis e computadores usados nos dias de hoje. Estes assistentes são compostos por dois sistemas principais, um de texto para voz e outro de voz para texto, que permitem interação usando linguagem natural, fornecendo assim uma forma mais fácil e intuitiva de comunicação entre o utilizador e a máquina.

Ao aperceber-se dos rápidos desenvolvimentos tecnológicos neste campo, a Talkdesk decidiu avançar com a criação do projeto Virtual Agent (Agente Virtual), cujo intuito é o desenvolvimento de um sistema capaz de responder a simples perguntas recorrentes em call-centers, permitindo assim que os agentes humanos se foquem em assuntos de maior complexidade, o que leva a uma otimização em termos de tempo e recursos.

Embora o objetivo inicial fosse o desenvolvimento de uma solução interna, a investigação realizada durante o primeiro semestre permitiu que a equipa adquirisse conhecimentos técnicos nesta área, o que por sua vez permitiu concluir que para alcançar melhoramentos sobre as implementação open-source existentes, seria necessário um investimento considerável em termos financeiros e temporais. Sendo assim, uma nova solução foi pensada.

Esta dissertação propõe o desenvolvimento de uma ferramenta interna de sintetização de fala, baseada em redes neuronais e aprendizagem profunda, para o módulo de texto para fala do projeto Virtual Agent, tendo o objetivo de detectar erros ocorridos durante a sintetização e permitir obter uma avaliação de vários sistemas de texto para fala de uma forma rápida e eficaz. A meta final é adquirir conhecimento sobre como um determinado sistema se comporta em relação aos áudio que gera, tendo em conta erros comuns como vozes robóticas ou a presença de longos excertos de silêncio no meio das frases.

Palavras-Chave

Agentes Virtuais, Texto para Fala, Redes Neuronais, Aprendizagem Profunda, Processamento de Linguagem Natural (NLP), Sintetização de Fala, Espectogramas, Deteção de Erros, Avaliação de Sistemas

This page is intentionally left blank.

Contents

1 Introduction **1**

2 Background Knowledge **6**
2.1 Machine Learning 6
2.1.1 Learning with Data 7
2.1.2 Problems and Algorithms 8
2.1.3 Algorithms 9
2.1.4 Neural Networks 10
2.1.5 Performance Metrics 14
2.1.6 Optimization Methods 16
2.2 Natural Language Processing 16
2.2.1 Phonology Phase 17
2.2.2 Morphology and Lexical Phase 17
2.2.3 Syntactic Phase 18
2.2.4 Semantic Phase 19
2.2.5 Discourse Phase 20
2.2.6 Pragmatic Phase 21
2.3 Speech Synthesis 21
2.3.1 Key Concepts 22
2.3.2 Sound Waves and Signal Processing Simplified 22
2.3.3 Approaches 25

3 State of the Art **29**
3.1 Existing implementations 29
3.1.1 Text-to-spectrogram 29
3.1.2 Vocoders 33
3.1.3 End-to-End 35
3.2 Evaluating Results 36
3.2.1 Automatic Evaluation 37
3.3 Datasets 39
3.4 Competition 41
3.5 Open-source implementations 42

4 Proposed Approach **45**
4.1 Requirement Analysis 46
4.1.1 Functional Requirements 46
4.1.2 Non-functional Requirements 49
4.2 Risk Analysis 50
4.2.1 Identified Risks 50
4.3 Methodology 52
4.4 Planning 54

4.4.1 First Semester 54
4.4.2 Second Semester 55
4.4.3 Full Internship 55

5 Development Approach **57**
5.1 Developed Work 58
5.2 Interrupted Development 59
5.3 Developed Classifier Models 59
5.4 Main Frameworks and Libraries 61

6 Experimentation and Results **65**
6.1 Classifier Summary and Overall Structure 65
6.2 Robotic Tone Classifiers 68
6.2.1 Binary Robotic Tone Classifier (B-RT) 69
6.2.2 3 Class Classifier for robotic tone (3MC-RT) 74
6.2.3 4 Class Classifier for Robotic Tone (4MC-RT) 79
6.3 Binary Classifier for Excessive Silence (B-S) 81
6.4 3 Class Mixed Classifier (3MC-MIX) 84
6.5 Final Overview 86

7 Conclusion **89**
7.1 Future Work 91

This page is intentionally left blank.

Acronyms

3MC-MIX 3 Class Mixed Classifier. x, xvi, xviii, 65, 66, 84–86

3MC-RT 3 Class Classifier for Robotic Tone. x, xv, xviii, 65, 66, 74–79, 84, 86

4MC-RT 4 Class Classifier for Robotic Tone. x, xv, xviii, 65, 66, 79, 80, 86

API Application Program Interface. 1, 4, 41, 42, 61

B-RT Binary Classifier for Robotic Tone. x, xviii, 65, 66, 69, 74, 81, 86

B-S Binary Classifier for Excessive Silence. x, xvi, xviii, 65, 66, 81–86

CNN Convolutional Neural Networks. 12, 13, 39

CPU Central Processing Unit. 54

dBFS Decibels relative to full scale. 25, 63, 82

F0 Fundamental Frequency. 22, 23, 30

FC Fully-Connected. 67, 70–72, 75–77, 80, 81, 83, 85, 86

FN False Negative. 15

FNN Feedforward Neural Networks. 12, 13

FP False Positive. 15

GPU Graphics Processing Unit. 2, 43, 51, 54, 91

GRU Gated Recurrent Units. 13, 29, 31, 32

KNN K-Nearest-Neighbors. 9

LSTM Long Short-term memory. xviii, 13, 32, 38, 66, 67, 70–72, 75–77, 80, 81, 83, 85, 86, 90

ML Machine Learning. 6, 8–10

MOS Mean Opinion Score. 36–38

NER Named Entity Recognition. 19, 20

NLP Natural Language Processing. 13, 16, 17, 19, 21

POS Part-of-Speech Tagging. 17, 18

ReLU Rectified Linear Unit. 11, 12

RNN Recurrent Neural Networks. 12, 13, 32, 33, 35

STT Speech-to-Text. 1, 17, 21, 43, 89

SVM Support Vector Machine. 10

TN True Negative. 15

TP True Positive. 15

TTS Text-to-Speech. 1–4, 17, 21–23, 25–27, 29, 31, 33, 36–38, 41–43, 45, 46, 57, 59, 65, 79, 87, 89–91

This page is intentionally left blank.

List of Figures

2.1 Overall dataset division(Shah, 2017). 7
2.2 Model Performance(Bhande, 2018). 7
2.3 Binary perceptron. 11
2.4 Basic Recurrent Neural Network (Olah, 2015). 12
2.5 Convolutional Neural Network (Saha, 2018). 14
2.6 The Transformer model architecture (Vaswani et al., 2017). 14
2.7 Learning rate effects (Jordan, 2018). 16
2.8 Dependency Parsing graph example(Jurafsky and Martin, 2019). 19
2.9 Continuous signal, represented by the green line, and a discrete represented by the various blue samples (*Sampling (signal processing)* n.d.). 23
2.10 A representation of a Spectrogram in Mel Frequency Scale(Gartzman, 2019). 24
2.11 Mel to Hertz Scale(*Mel Scale* n.d.). 24
2.12 The overall TTS architecture evolution (X. Wang, 2019). 26

3.1 Original Deepvoice architecture (Arik et al., 2017). 30
3.2 Multi-Speaker Deepvoice 2 Architecture (Gibiansky et al., 2017). 31
3.3 Original Tacotron architecture(Y. Wang, Skerry-Ryan, et al., 2017). 32
3.4 Original FastSpeech architecture and FFT block (Ren et al., 2019) 33
3.5 Overview of Probability Density Distillation (Oord, Y. Li, et al., 2017). 34
3.6 Overview of Clarinet Architecture (Ping, Peng, and J. Chen, 2018). 36
3.7 AutoMOS system architecture (Patton et al., 2016). 38
3.8 MOSnet system architecture (Lo et al., 2019). 39
3.9 LibriSpeech Overall Structure. 40

4.1 Gantt chart for the first semester. 54
4.2 Gantt chart for the second semester. 55
4.3 Gantt chart for the full internship. 55

6.1 Scatter plot of audio lengths (in seconds) distribution. 68
6.2 Spectrograms depicting the original, delta, delta delta and concatenated representations of a Tacotron audio. 68
6.3 AutoMOS system architecture, already seen in 3.7 but duplicated here to improve reading flow. 70
6.4 AutoMOS Network Tensorflow interpretation and implementation. 71
6.5 Graphical Confusion matrix and evolution of accuracy and loss across epochs for the 60 Mel binary model. 73
6.6 Spectrograms for the various 6 original sources. 74
6.7 Graphical Confusion Matrix for 3 Class Classifier for Robotic Tone (3MC-RT) model 2.1. 78
6.8 Accuracy and Loss for 3MC-RT model 2.1 78
6.9 Graphical Confusion matrixes for four class 4 Class Classifier for Robotic Tone (4MC-RT) model 1 and 4MC-RT model 2. 80

6.10 Energy and Spectrogram representation of a signal suffering from excessive silence. 82
6.11 Graphical Confusion matrix and evolution of accuracy and loss across epochs for Binary Classifier for Excessive Silence (B-S) model 2.1. 83
6.12 Best performing network for B-S 84
6.13 Graphical Confusion Matrix for 3 Class Mixed Classifier (3MC-MIX) Model 1 86

This page is intentionally left blank.

List of Tables

2.1 Caption for LOF 20

3.1 The Mean Opinion Score ratings. 37
3.2 Overall state of the art systems comparison. 37
3.3 Overall best hyperparameters for AutoMOS 39
3.4 LJ Speech Structure. 41
3.5 Mean Opinion Score ratings. 42

5.1 Functional Requirements Status. 58
5.2 Example of metrics taken for a single utterance. 60
5.3 Example of metrics taken for a single dialog. 60

6.1 Developed classifiers overview. 66
6.2 Available hyperparameter, their description and presence on the original AutoMOS paper. 67
6.3 Robotic audio dataset used for the Binary Classifier for Robotic Tone (B-RT). 69
6.4 Optimized hyperparameters values and corresponding phase. 70
6.5 Relationship between number of Long Short-term memory (LSTM) units and possible fully connected layers configuration. 71
6.6 Best architectures for each number of mels tested. Note that the 60 mels model is the best performing one. 72
6.7 Numerical Confusion Matrix of the 60 mels robotic model. 73
6.8 First approach to 3MC-RT labels. 75
6.9 Initial 3 best models for 3MC-RT 75
6.10 Final dataset for 3MC-RT. 76
6.11 5 best models for 3MC-RT, including models with 50 samples for each class. 76
6.12 Final 5 best models for our 3MC-RT, after excluding the models where the dataset size was 50. 77
6.13 Numerical Confusion Matrix for 3MC-RT model 2.1. 78
6.14 Dataset used for representing 4 robotic classes. 79
6.15 3 best models for our 4MC-RT. 80
6.16 Numerical Confusion Matrix for 4MC-RT model 1. 80
6.17 Excessive Silence dataset initial approach. 81
6.18 First iteration of 3 best models for B-S. 81
6.19 Improved silence dataset. 82
6.20 Second iteration of 3 best models for B-S after improving the dataset labels. 83
6.21 Numerical Confusion Matrix for B-S model 2.1. 83
6.22 Dataset used for representing the 3 classes for 3MC-MIX. 85
6.23 3 best models for 3MC-MIX. 85
6.24 Numerical Confusion Matrix for 3MC-MIX Model 1. 85
6.25 Comparison of the best model architecture for each classifier. 86

This page is intentionally left blank.

Chapter 1

Introduction

The development of systems capable of producing sounds with the goal of synthesizing speech, dates back to the *XVIII* century. While these initial models were barely producing the sound of vowels, current models are capable of generating fluent speech in a variety of languages, different voices and even with specific intonation. State-of-the-art approaches to speech synthesis use neural networks and deep learning, making use of trained models to convert text into speech, with the most commonly known solutions being the ones from giant technological companies such as Google[1], Amazon[2], Microsoft[3] and Apple[4]. These companies integrate their Speech-to-Text (STT) and Text-to-Speech (TTS) systems in order to create virtual assistants, a feature present in most current smartphones and computers.

Talkdesk[5] saw these latest and quick technological advances as an opportunity, taking advantage of them to start the Virtual Agent project. The Virtual Agent aims to help call-center callers with simple and repetitive questions, such as reseting a password, allowing human agents to focus on more complex matters and, as a consequence, optimizing their time. In order to achieve it, a conjunction between TTS and STT systems must be made, allowing the virtual agent to interact with a real person as if it was a human agent.

This work will have a special focus on Virtual Agent's TTS component, studying its evolution, current implementations, flaws and potential upgrades. While great quality TTS Application Program Interface (API)s are currently available and would be able to fulfill this task, they are often expensive when used in a large scale professional environment and are difficult, or impossible, to modify and personalize if the need arise. This is the reason why we have the goal of developing a more sustainable, high-quality, in-house solution with the final objective of being fully integrated with the Virtual Agent project.

During the first semester, the aim was mainly:

- Search and study the already existing solutions for TTS, the technology behind them and how it was applied.
- Use the available open-source solutions in public repositories and compare them in terms of results and performance.

[1]https://www.google.com
[2]https://www.amazon.com
[3]https://www.microsoft.com
[4]https://www.apple.com
[5]https://www.talkdesk.com/

- Start the development of a proof of concept.

However, as a result of our studies, we soon found out that in order to increase the performance of these solutions beyond the already provided, extensively trained models, the creation of much better datasets with crystal clear sound would be needed to train new models, which would require high-end machines with various Graphics Processing Unit (GPU)s, to be achievable in a realistic timeframe. To summarize, we initially underestimated the resources needed for experimentations, even after searching for various alternatives.

With this said, the initial internship's goal changed across the time, with two main changes taking place across the two semesters. The first change was going from the development of an in-house TTS solution to an error detection system, while the second change consisted in modifying the latter idea and develop a solution that would allow the evaluation of a given speech synthesis system, taking into account some of the more common errors during synthesization. Overall, the path made in this project can be divided into three distinct phases.

First Phase

The first phase of this work consisted in the initial idea of developing an in-house neural-network based TTS solution from the ground up, which would avoid the company having to rely on third-party systems with their own costs and development schedules.

The first step was researching on the existing solutions and methods used to develop them, leading to the discovery of the main state of the art systems. In order to better understand the complex architectures of these systems, a deep-dive on various topics of machine learning and neural networks also had to be done.

While researching, various open-source TTS implementations were found, developed using some of the state-of-the-art algorithms, with some of these providing actual pre-trained models allowing to completely skip the training phase while achieving surprisingly good results. However, as with any system, flaws were found on most implementations which led to the search of how they could be improved by modifying the existent code, datasets and retraining the existing models. It did not take long for three main problems to appear:

- The considerable time and financial resources needed to retrain these models.
- The sheer complexity of the used algorithms.
- The uncertainty in how these already high quality datasets could be further improved.

The first and second problems are mostly correlated as, due to the complexity, there was no certainty of achieving improvements with every modification and retrain. Furthermore, the amount of time spent and hardware needed for training was completely out of the current possibilities. After some calculations, it was concluded that the investment needed for such a low degree of certainty would be unrealistic.

Finally, regarding the third problem, improvements to the the datasets would need to be made with professional recorded audio and even then it was not ideal to use different voices, as at the time the aim was at a single-speaker system with higher quality instead of a multi-speaker one.

Although various attempts were made in order to avoid these problems, they were mostly unsuccessful. The degree of uncertainty on achieving improved results was too high, making them unrealistic given the required costs.

With this in mind, one possible solution was externally improving the systems without re-training, which led us to our second development phase.

Second Phase

Taking into consideration what was learnt on the first phase, we started planning on the development of a system capable of identifying and solving the main problems of the existing open-source solutions during real-time started. This project would allow improvements on these systems by solving some of their expected problems in a production environment during runtime, without having to rely on retraining the existing models. Additionally we would also add elements for making the system more real, such as background office noise, making the interaction more realistic.

After designing some possible architectures, it became evident this was not the best approach for improving the existing systems, since building a production ready system with the planned characteristics while also achieving good performance was overwhelming and unrealistic on the given internship time-frame. With this in mind, it was decided the project goal would slightly change one final time, leading to the third and final phase.

Third Phase

The third phase was the last iteration on the project's direction, being the one where most of the practical development was done. After going through the previous phases, there was a clearer vision on the overall challenges that could be faced. However, one of the questions that persisted was *"Which is the best model for Talkdesk's TTS system?"*. This is not an easy question as each system has advantages over the others.

As an example, a given model is extremely quick during inference but lacks quality, suffering from robotic audio in most cases. On the other hand, other model is considerably slow but outputs very clean speech. Comparing systems proved to be a cumbersome and time consuming task, which led to thinking in ways of automatizing the process.

The goal was then set in developing one or more classifiers that, given a large amount of audios, would classify each of them according to previously identified errors. This would be useful in quickly understanding if a given system generated more clean audios than robotic ones or even the amount of audios with excessive silence or critical errors. This would drastically reduce the time needed to have an overall evaluation of a TTS system as it was no longer a necessity to manually hear a large amount of generated samples, by quickly being able to see an overview on the performance of the given system.

This document will mainly focus on the work done in the first and third phases, since they mostly correspond to the research and practical parts of the internship respectively. The document is organized as follows:

- **Chapter 2: Background Knowledge -** The second chapter of this document aims at giving readers the necessary knowledge to understand our work, with a brief introduction to various concepts regarding machine learning, natural language processing, neural networks and speech synthesis.
- **Chapter 3: State-of-the-art -** Our third chapter introduces the current existing

state-of-the-art approaches regarding TTS systems and techniques used when evaluating the output of these implementations. It also gives an overview of famous voice datasets used for training these models and who is the current competition in terms of public API's for speech synthesis.

- **Chapter 4: Methodology and Planning -** The fourth chapter gives an overview of the tools and processes used at Talkdesk's Virtual Agent team during the internship, the requirements and risks of the projects and the overall project's schedule during the first and second semester.
- **Chapter 5: Development and Tools -** The fifth chapter is the shortest one, with the aim of showing which requirements were fulfilled during the internship as well as some frameworks and libraries used for development of certain features.
- **Chapter 6: Experimentations and Results -** The sixth chapter is aimed at showing all the experimentation process and associated results. However, since this work relies heavily on these aspects, it made sense to also include most of the information regarding the developed classifiers and decisions here instead of the previous chapter.
- **Chapter 7: Conclusion -** The last chapter consist on an overview of the work done as well as a proposal for future work regarding the project.

This page is intentionally left blank.

Chapter 2

Background Knowledge

In this chapter, we aim at giving basic knowledge about the topics that are in the basis of this work, from the definition of what is Machine Learning (ML), how we use data to train models, some techniques to process text to some background on neural networks as well as some approaches.

2.1 Machine Learning

By definition, ML (YourDictionary, 2019) is the field of study concerned with the design and development of algorithms and techniques that allow computers to learn. Essentially, it is a software that modifies and adapts itself by using example data or experience.

In computer science, algorithms can be seen as a means to achieve a solution when trying to solve a given problem, a set of instructions given to a computer in order to transform an input into the desired output. However, what happens when the input changes with time or with a given environment? A static, linear algorithm will start to struggle, as the input it was originally expecting is no longer the one being received.

As a simple and common example, let's take a look at spam emails. To identify and filter spam emails, algorithms must look at certain suspicious aspects of a given email such as keywords, phrases and language, among others, to classify the email as spam or not spam. This signifies they are expecting certain words or expressions as an input, and by detecting them they will be able to classify the email. However, as soon as the spammers (people or system who sends spam emails) realize how their emails are being filtered, they will change them. After this change, a linear algorithm would stop being successful at filtering as the expected input had changed. This is where ML starts to shine. ML algorithms use past and current data and experience to learn what is or isn't a spam email even after the changes, in order to make predictions and automatically develop methods to catch them. They evolve themselves to expect a different input and the more data we feed these type of algorithms the better will those predictions be (Alpaydin, 2009).

In this section, a brief introduction will be made in how data can be used to learn or to develop a system, the main ML problems, a description of the more common algorithms and an overview of neural networks.

2.1.1 Learning with Data

Compilation of data from past experiences is called a dataset. These datasets are usually divided into training, validation and testing.

In the training phase, the model sees and learns from the full spectrum of information in order to adjust its parameters and improve in performance. The validation phase is an optional phase, used to fine tune the model, by still seeing all the information but no longer adjusting its internal weights. The developer, however, should use the validation results to adjust the high level parameters by hand. Finally, the testing phase is used to evaluate the results of the model. It is seen as the gold standard, only used once the model training is complete. There isn't a set rule in how the original dataset should be divided in order to have these three different sets. It is important, however, to give the train set the biggest size. The overall dataset division can be seen in figure 2.1.

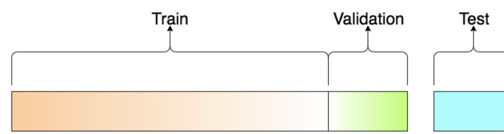


Figure 2.1: Overall dataset division(Shah, 2017).

The size of each set can be related to under-fitting, where a model is underperforming, and over-fitting, where a model is over-performing, which means it is too well adapted to the given training set and will probably have bad performance when presented with unseen information. Figure 2.2 illustrate well these scenarios, where appropriate-fitting should be the objective when tuning the final model.

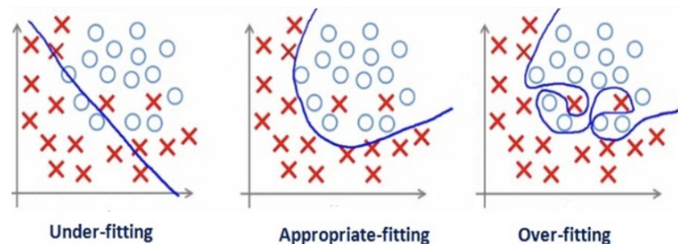


Figure 2.2: Model Performance(Bhande, 2018).

Instead of doing a fixed division of the dataset, a technique called k-fold cross-validation can also be used, where the whole dataset is split into k sets and each of this sets is used as the test set in k different runs. As an example, if $k = 5$, we would have 5 groups with 80% of the data for training and 20% for validation. 5 models would then be trained and validated, each using a different group. After all the runs, the mean of the results should be taken in order to understand how our model is performing. This is especially useful when the dataset is small, as we are making use of all the data (Alpaydin, 2009).

It is common to go through the dataset information more than one time in iterative learning. Each one of these iterations is called an epoch. So, for example, in 200 epochs the training dataset went trough the learning algorithm 200 times. Epochs are an important parameters as it allows us to avoid under-fitting. However, when in excess, they will start to cause over-fitting. Overall, under-fitting and over-fitting can be seen as a trade-off between generalization and specialization on the training database.

Another technique used is batched and unbatched training. In unbatched training, the whole dataset is fed to the neural network at once, while in the batched method the training dataset is divided in X smaller parts and fed separately in various iterations. In this case, an epoch occurs every X iterations.

Datasets can contain labeled data or unlabeled data. Using the previous example, a labeled spam email dataset would contain entries of email text and the target label (also called tag or class) stating if each entry should be considered spam or not. Labeling data is usually done by humans, which as a consequence, makes labeled datasets more scarce than unlabeled datasets. In this last type of dataset, the categorization of each entry is yet to be done. However, ML algorithms can also make use of unlabeled (or raw) data by applying specifics techniques in the classification phase, which we will explain later.

With this said, there are three main learning methods: supervised, unsupervised and reinforced learning (Alpaydin, 2009).

Unsupervised Learning

In Unsupervised Learning (Alpaydin, 2009) there is only raw data and, as such, the aim is to find similarities and patterns in order to classify each entry. The most common method to achieve this goal is clustering, where similar entries are grouped into cluster and each cluster is considered a different class. We will dive deeper on clustering in the next section.

Supervised Learning

Supervised learning (Alpaydin, 2009; Bishop, 2006), as the name suggests, use labeled datasets. Due to the existing labels, it is already known to which class does a group of characteristics (features) belong, so each data entry is already part of a certain class. While training and learning from the given labeled data, the model will adjust its internal parameters in order to predict future outputs.

Reinforcement Learning

Reinforcement Learning (Yiu, 2019) is based on rewarding or penalizing the actions of an agent, whose responsibility is exploring an unknown environment, always having an end goal. In other words, it can be seen as unsupervised learning with an end goal defined. Actions that bring the agent closer to the end goal bring a positive reward, while getting farther away from the goal will bring negative rewards.

This learning approach is especially useful when previous data does not exist, as an agent is essentially learning and building experience by trial and error.

2.1.2 Problems and Algorithms

As we just saw, labeling data is an essential task in ML in order to achieve results, but different learning methods and types of data require different labeling techniques. Overall there are four class of problems in ML, *classification, regression, clustering* and *association* (Bishop, 2006).

Classification and Regression

Classification (Bishop, 2006) is one of the problems when supervised learning is being used, where the main goal is to classify, or label, an entity into a certain class, always taking into account the characteristics, more commonly referred to as features, of said entity. The number of classes and type of features are defined in the input data, with the algorithm knowing them beforehand by having access to the dataset. The algorithms tackling this type of problems are usually known as classifiers.

Regression is another type of problem for dealing with data in supervised learning. While classification outputs a given class or label, such as "spam" or "not spam", for each entity, regression consists in the attribution of a real or continuous value given the input features. As an example, we would need to use regression when trying to predict stock prices.

In simple terms, classification final output is categorical while regression output is numerical.

Clustering and Association

Clustering (Bishop, 2006; Wagstaff et al., 2001; D. Xu and Tian, 2015) is a technique that divides the available data into groups, referred to as clusters, where each cluster has elements with similar characteristics and features. It is mostly used on unsupervised learning problems, as it assumes there is no prior knowledge on the classification of the elements in the dataset, although it can also be used when there is more background knowledge.

Association (Harrington, 2012) problems rely on finding rules and relations that are capable of describing a big percentage of our data, in order to find patterns and identify new data points. These relationships have mainly two forms, frequent items sets with entities that usually occur together, and association, when the probability of strong relationship between two entities is high.

2.1.3 Algorithms

In order to solve the problems introduced in the previous section, ML algorithms are used. There are no best or worse algorithms and when solving a given problem, more than one algorithm should be used and tested, in order to find the one with greater performance for the given situation. The most commonly used algorithms are the following:

- **Naive Bayes -** Based on Bayes Theorem, which states that the probability of an event is conditioned by our prior knowledge of conditions related to that event, the Naive Bayes algorithm assumes that the presence of a particular feature in a given class is completely unrelated and independent to the presence of another, hence the name naive (Alpaydin, 2009).

- **Logistic Regression -** Logistic Regression (Gandhi, 2018) uses a linear function to perform classification, with the outputs going through the sigmoid equation (Logistic Function), which we will see in a future section, converting the output of our algorithm in a 0 or 1. When misclassification occur, the cost is calculated using the logarithmic loss function.

- **K-Nearest-Neighbors (KNN)-** KNN (Harrison, 2018) is based on proximity, where the closest entities around the point we are trying to classify are taken into account. The algorithm search the K closest entities of a certain point, finds the

mean of those points and classifies our point according to that information. As an example, if we were trying to classify emails with $K = 5$ and had a given point whose closest neighbors were 3 spam entities and 2 non-spam entities, that point would be classified as spam. For regression problems, the value of our point would be the mean of the closest 5. In order to find the right value of K, experimentation must be done as a very low or very high K will be more susceptible to classification errors. In addition, K should also be an odd number, to avoid problems such as ties (for example, 3 spam entities and 3 non-spam entities around our point).

- **Support Vector Machine (SVM) -** SVMs (Pupale, 2018) aim to create a line (for binary classification) or hyperplane (for multi-class classification) between points in order to define to which class those points belong. The goal is to maximize the distance, referred to as margin, between the line or hyperplane and the support vectors. The support vectors are the closest points from the various classes that are closer to the margin. The bigger the margin, the better, as there is less chance of a wrong classification. When a given problem is not linearly separable, the data must be converted by, for example, adding one more dimension.

- **Random Forest -** Random Forest (Yiu, 2019) is an agglomeration of decision trees, a simpler type of algorithm where decisions are made in a tree-shaped architecture where each branch represents a decision and the leaves represent all the possible outcomes. Each tree of our forest is independent from the others, where the final classification of the given input is commonly referred to as a vote. The results are then compared and the class with more votes is the one assigned to our input.

- **K-Means -** K-means is used to perform clustering, where the goal is to partition the existing data into K clusters (and, as such, dividing the data into K different classes). Initially, the algorithms chooses the k initial cluster centers, called centroids, and optimizes them interactively with each interaction consisting on the following simplified steps:

  1. Compute the distance d_i from each entity and the closest cluster center.
  2. Adjust the centroid computing the mean of all its entities.

  Convergence is achieved when there are no more adjustments to be made. The distance between elements and centroids can be computed using different methods, with the most common being the Euclidean and the Cosine distance (Bishop, 2006; Wagstaff et al., 2001; D. Xu and Tian, 2015).

- **Apriori Algorithm -** One of the main algorithms in association problems, the Apriori Algorithm (Harrington, 2012) pairs entities in order to see how strong is their relationship. Pairs with weak relationships are tossed out, while the others are kept.

2.1.4 Neural Networks

ML's neural networks (Nielsen, 2015) take inspiration from the human brain where neurons, the brain single units, are connected to each other and transmit electric signals when they receive a strong enough stimulus.

It can be said that artificial neural networks started with the perceptron, seen in figure 2.3, a simple system with binary inputs and a single binary output. In order to know the output, inputs are multiplied by real numbers called weights, which express the importance

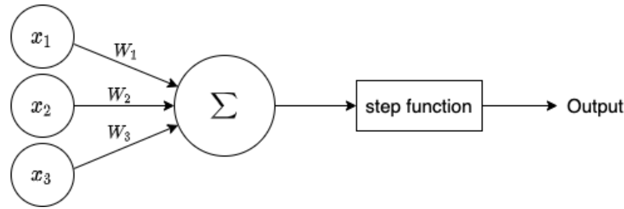


Figure 2.3: Binary perceptron.

of each input. If the weighted sum is bigger than given real number threshold, the output is 1 and if its smaller it is 0 as seen in equation 2.1.

$$output = \begin{cases} 0, & \text{if } \sum_j x_j w_j \leq threshold \\ 1, & \text{if } \sum_j x_j w_j > threshold \end{cases} \tag{2.1}$$

In addition to the inputs and weights, it is usual to mention another variable with the name of *Bias*, which can be seen as a measure of how easy it is to fire the perceptron, obtaining the output with value 1. As we can see in equation 2.2, as the Bias value b increases, the easier it is to reach 1 as the final output. Looking at both equations it is easy to understand they are equivalent, with 2.1 using a vector multiplication instead of the dot product and $b = -threshold$. In terms of overall notation, using equation 2.2 is preferred over equation 2.1.

$$output = \begin{cases} 0, & \text{if } x \cdot w + b \leq 0 \\ 1, & \text{if } x \cdot w + b > 0 \end{cases} \tag{2.2}$$

However, this binary approach proved to have some flaws, such as the XOR problem (Ahire, 2017). Improving upon these problems, perceptrons were changed to allow inputs and outputs between $[0, 1]$ instead of being only binary. In order to calculate the value of the output, the sigmoid function, see equation 2.3, was used. Using the name of this function, the new approach was called "Sigmoid neuron", being the base of all the current neural networks (Nielsen, 2015).

$$output = \frac{1}{1 + e^{-x}} \tag{2.3}$$

Currently there are alternatives to the sigmoid function such as Softmax, a function that returns a probability distribution with a sum of 1 (essentially replacing sigmoid in multi-class problems, as long as classes are exclusive) and the hyperbolic tangent activation function (Tahn), see equation 2.4, a resized sigmoid function with a more convenient range of [-1,1].

$$output = \frac{e^x - e^{-x}}{e^x + e^{-x}} = 2 * sigmoid(2x) - 1 \tag{2.4}$$

These functions, however, do have problems when applied to very deep networks, due to the vanishing/exploding gradient problem, where the transmitted error across layers tends to vanish, causing the weights to stop being updated. In order to avoid this problem, the Rectified Linear Unit (ReLU) function is used. It is a non-linear function, working in a range of $[0, \infty]$, where negative numbers are immediately assigned a value of 0 as equation

2.5 shows, although slight alternatives such as the leaky-ReLU change this range. A good comparison can be seen in (B. Xu et al., 2015).

$$output = \begin{cases} 0, & \text{if } x_i \geq 0 \\ x_i, & \text{if } x_i < 0 \end{cases} \tag{2.5}$$

Until now we only talked about isolated neurons, but in most cases they are actually grouped. A group of neurons working in parallel is referred to as a layer, with a network being capable of having multiple layers one after the other. The first layer has the name of input layer, the last layer the output layer and in-between are the hidden layers, where the input of each one is the output of the previous one. Networks with only one hidden layer are commonly referred with the terms shallow networks and shallow learning, where networks with various hidden layers are referred to with terms as deep networks and deep learning (Nielsen, 2015).

There are various implementations of neural networks, each with its own advantages and disadvantages. In the next paragraphs we will describe some different implementations, such as Feedforward Neural Networks (FNN), Recurrent Neural Networks (RNN) and Convolutional Neural Networks (CNN).

Feedforward Neural Networks

FNN (Nielsen, 2015) are the most common and basic type of neural networks, composed by an aggregation of neurons where the information only flows in one direction, forward, hence the name feedforward. A multilayer perceptron is considered to be a FNN, where we receive the inputs, give them weights, pass them to the hidden layers and, finally, compute the final output.

While FNN are relatively simple and useful in a variety of scenarios, they do prove to have some problems when context matters, as the single flow of information does not provide memory regarding past inputs.

Recurrent Neural Networks

RNNs can be seen as an improvement of FNN, having the capability of handling a variable-length sequential input due to having an hidden state. This hidden state can be seen as a feedback loop, where an output also becomes an input. In other words, each neuron has two inputs, the normal one and one with the previously outputted information, allowing it to have memory of almost all the information across time.

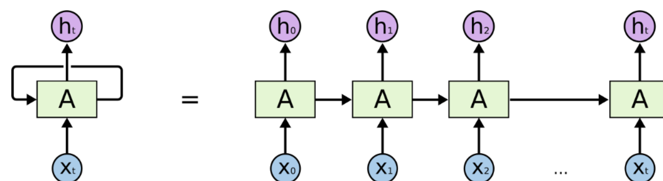


Figure 2.4: Basic Recurrent Neural Network (Olah, 2015).

Figure 2.4 illustrates how each previous output is fed to the next input in a simple RNN architecture, where A stands for the neural network, x_t for each input and h_t for each output. This is especially useful when context is important during decision making. The

most common example is text, where the meaning of a word usually depends on the context of the phrase. However, RNNs suffer from short-term memory, having problems retaining information in lengthy inputs. This is usually referred to as the vanishing gradient problem and happens during backpropagation (Rumelhart et al., 1986).

To tackle this issue, Long Short-term memory (LSTM) networks were developed in 1997 (Hochreiter and Schmidhuber, 1997) and Gated Recurrent Units (GRU) in 2014 (Cho et al., 2014).

LSTM's introduce a memory cell with three gates in order to control the flow of information. An input gate reads data into the cell, an output gate reads data from the cell and a forget gate resets the content of the cell. With this mechanism, LSTM's are able to decide whether to keep existing memory or not and can easily carry information from early stages, hence capturing potential long-distance dependencies. Improvements upon the original LSTM's have been as can be seen in (Chung et al., 2014).

GRUs only have two gates, a reset gate and an update gate, ditching the memory cell state. They are seen as a quicker version of LSTM's, however it does not always translates into improved results (Chung et al., 2014).

Convolutional Networks

CNN (LeCun et al., 1998) are a special type of FNNs, inspired by the brain's visual cortex organization and characterized for having a varied depth and breadth, two variables that control their capacity. When compared to simple FNNs, they are easier to train, having fewer connections and parameters while also achieving similar levels of performance and being especially good at image analysis, where a FNN would, more often than not, struggle due to the amount of information (Krizhevsky et al., 2012).

CNNs apply filters (called Kernels) with shared weights to the input in order to simplify it and extract the highest level features (for example, the edges of an image). This operation is called the convolution operation and can consist of one or more convolutional layers, where each layer applies convolutions to the output of the previous one. The pooling layer comes after all the convolutional layers and has the objective of reducing spatial size, by applying dimensionality reduction which has the consequence of reducing the needed computational power. Finally, the resulting output of the previous layers, a considerably simplified version of the original input, will be flattened into a column vector and fed to a normal FNN. Figure 2.5 represents how the whole process is connected and how layers can be repeated for improved simplicity. While CNN's were originally associated with images, they are becoming extremely useful when dealing with Natural Language Processing (NLP) tasks (Kalchbrenner et al., 2018).

Attention and Transformers

Attention mechanisms (Bahdanau et al., 2014) are an important module in deep learning as they greatly improve the context awareness of our neural networks. First introduced with the aim of translation problems, attention tries to find the most relevant information in a text sentence by encoding each word into a hidden state, instead of encoding the whole sentence at once, and using weights for each hidden state. Self-Attention is a technique that also looks at other words in the sentence before converting a given word to the hidden state. Attention mechanisms were first introduced in order to improve RNNs and LSTMs.

To allow parallelization, Transformers (Vaswani et al., 2017) were introduced, being essen-

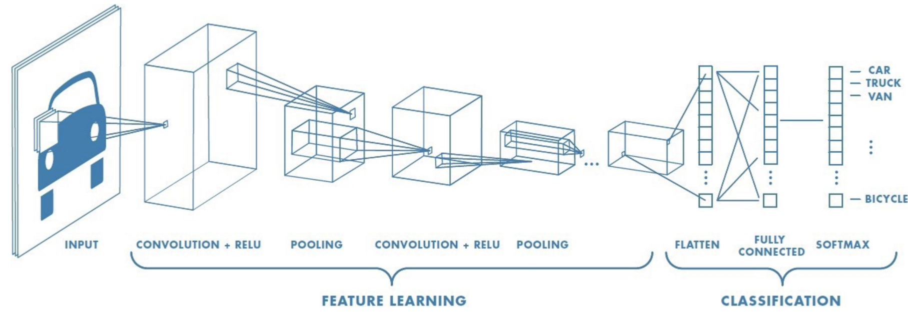


Figure 2.5: Convolutional Neural Network (Saha, 2018).

tially composed of 6 encoders and 6 decoders. Each of the encoders is composed of two sub-layers, a multi-head self attention and a feed-forward network. The decoders adds an additional layer besides the ones in the encoder, where multi-headed attention is performed on top of the output from the encoder. The decoder's multi-head attention layers is also slightly modified.

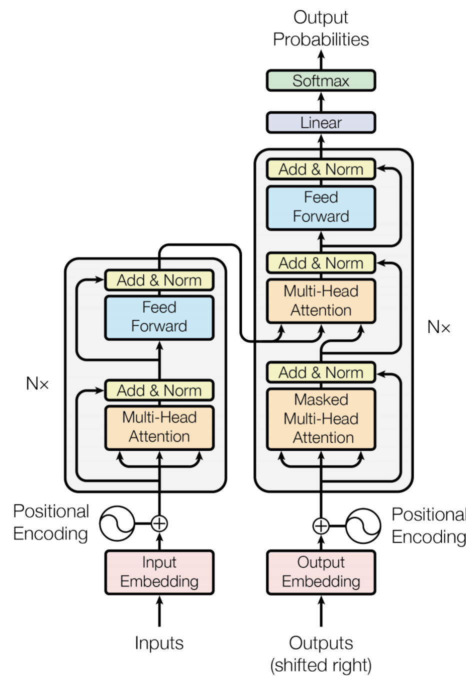


Figure 2.6: The Transformer model architecture (Vaswani et al., 2017).

As we can see in figure 2.6, the output of the decoder is fed into the linear and softmax layers. In these layers, the output is transformed into a vector of probabilities. In a translation problem, each word is associated with one of these probabilities, where the transcription is assumed to be the most probable word.

2.1.5 Performance Metrics

In order to evaluate the results of our models, evaluation metrics are needed. Ideally, more than one metric should be used, as a high score in one does not directly imply a good

overall performance of our model (Mishra, 2018; Sunasra, 2017).

Some of these metrics make use of a confusion matrix, an intuitive method which helps us organizing output data. In a binary example, there are four main terms used in this method, *True Positive (TP)*, when the predicted class is correct with a value of 1, *True Negative (TN)* when the predicted class is correct with a value of 0, *False Positive (FP)* when the classification predicts 1 but the expected value was 0 and *False Negative (FN)* when 0 was predicted but 1 was expected. In multi-class problems, the matrix can be built with the expected values as columns and the predicted values was rows, where all the correctly predicted values will be on the matrix diagonal(Mishra, 2018; Sunasra, 2017).

The main machine learning metrics are the following:

- **Classification Accuracy -** More commonly referred as simply accuracy, is the ration of correct predictions and the total number of input samples. It is well suited if there are an equal number of samples of each class. However, it does not take into account the cost of misclassification, which is a major flaw. Making use of the confusion matrix, it is the ration between the sum of TP and TN and the sum of all elements the matrix elements using $\frac{TP+TN}{TP+TN+FP+FN}$.

- **Precision -** Precision tells us how precise our model was in predicting a certain class. If we want to know the precision of TP, we would need to find the ratio between TP and the sum of TP and FP using $\frac{TP}{TP+FP}$.

- **Recall or Sensitivity -** Recall tells us the number of cases a given class was predicted, across all the entities of that class. For example, we would find the ratio of TP and the sum of TP and FN using $\frac{TP}{TP+FN}$.

- **Specificity -** Specificity is the exact opposite of recall, telling us how many negative classes were predicted. To do this, the ratio of TN and the sum of TN and FP would have to be calculated with $\frac{TN}{TN+FP}$.

- **F1 Score -** F1 Score aims to represent both precision and recall, making use of the harmonic mean, represented by $\frac{2xy}{x+y}$ or, replacing the variables, $\frac{2*Precision*Recall}{Precision+Recall}$. By using this formula, the final performance metric will be much closer to the smaller value, avoiding potential errors.

- **Logarithmic Loss -** Log Loss is suited for multi-class classification, where each class has a given probability of occurring. If misclassification occurs, a penalization is done. The higher the log loss, the worse is the performance of our model. Log loss is also called cross-entropy.

- **Area Under Curve -** Area under the ROC curve is the area under the curve in the plot of Specificity vs Sensitivity. The higher the area, the better as more True Positives were predicted.

- **Mean Absolute Error -** This metric obtains the average of the difference between original and predicted values, passing the information of how far were the predictions.

- **Mean Squared Error -** Mean squared error is similar to Mean Absolute Error but uses the square of the difference in order to improve computation in terms of gradient. In this approach, larger errors are more evident.

2.1.6 Optimization Methods

We have previously seen that loss can be used to understand how well a given network or algorithm is performing. As a rule of thumb, we should always aim at lowering our loss while, simultaneously, increasing our accuracy. The loss output of our loss function is used by the chosen optimization algorithm to update and improve our model.

One of the most popular optimization methods is the Gradient Descent, where the effect on the loss of changing a weight is calculated. The aim is to find the minimum loss for a given weight. However, this is susceptible to local minima, where we could think the algorithm has achieved the lowest value of loss possible but it has simply found a local minimum.

The degree of change of a given weight at each step is given by what is called Learning Rate. This is usually a small number, such as 0.001, although it is highly dependent on the problem and various values and scales should be tested. Ideally, a big learning rate is not desirable, as big jumps are going to be made, while very small rates lead to very small increments and computational time, increasing the probability of finding and getting stuck in local minima. Figure 2.7 illustrates this situation.

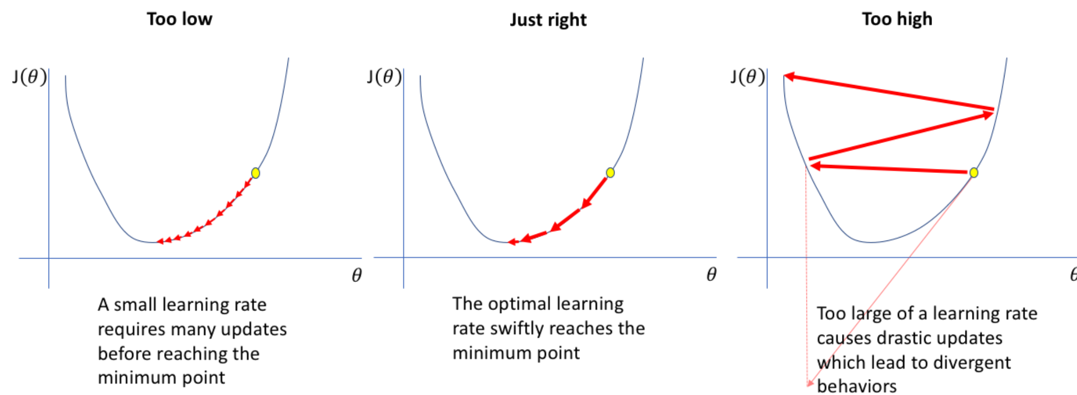


Figure 2.7: Learning rate effects (Jordan, 2018).

In this thesis, two optimizers were mainly used, both being based on gradient descent optimization. Adagrad (Duchi et al., 2011), as it was the one used on the AutoMOS paper and Adams (Diederik P Kingma and Ba, 2014), a more recent and popular approach that also leverages the advantages of Adagrad.

Regularization is used to avoid some weights to grow much bigger than the others, dominating the overall formula of our model. By applying regularization, a penalty is being made to big weights. In this work we will use two types of regularization, namely L1 (Lasso Regression) and L2 (Ridge Regression) (Gupta, 2017).

2.2 Natural Language Processing

NLP (Chowdhury, 2003) is a subfield of various disciplines such as computer and information sciences, linguistics, artificial intelligence and mathematics, aiming at exploring how computers can be used to understand and use natural language in order to perform desired tasks. The main objective of NLP is allowing humans to interact with machines verbally, or textually, in the same way they would interact with other humans. In other words, the

aim is to surpass what is referred to as the Touring test, where a human can't distinguish if an interaction is being made with a machine or with a person (Turing, 2009). One of biggest challenges in NLP is ambiguity, where words, phrases or punctuation are capable of having different meanings.

Ambiguity is tackled in different ways, according to each phase of the overall NLP pipeline. NLP problems are usually divided into sub-problems, where each sub-problem is treated in a different phase, or level, better known as the linguistic levels of knowledge. While the number of distinct phases may vary between authors, the most commonly referred are the phonology phase, the morphology phase and lexical phase, the syntactic phase, the semantic phase, the discourse phase and the pragmatic phase. We will delve into each one in the following sections.

2.2.1 Phonology Phase

Phonetic analysis (Richards and Schmidt, 2013) is the study of how sounds are grouped in order to form words. We will enter in greater detail about phonemes in the next section. The phonology phase is mostly present in Speech-to-Text (STT) or Text-to-Speech (TTS) applications, as phonology refers to sound. NLP tasks whose aim only deals with written text usually have little need for this specific phase.

2.2.2 Morphology and Lexical Phase

A morpheme is the smallest, meaningful unit in a given language. While some words consists of only one morpheme, such as *kind*, others consist of more than one such as *unkindness*, that consist of three with a prefix (*un*) and a suffix (*ness* in addition to the base one (Richards and Schmidt, 2013).

In the morphology and lexical phases the input text is segmented into words, phrases and paragraphs. Lexemes, the smallest unit in the meaning system of a language, are identified and converted to their base form to simplify the process. Inflected words such as *give*, *gives* and *given* are considered to belong to the same lexeme *give*(Richards and Schmidt, 2013). After the segmentation, Part-of-Speech Tagging (POS) occurs, where words are classified as nouns, verbs, pronouns, prepositions, adverbs, conjunctions, participles and articles (Jurafsky and Martin, 2019). This phase has challenges related to segmentation, which we will discuss further, and to lexical ambiguity. As an example, the word *silver* can be considered a verb, noun or adjective.

Tokenization

Tokenization (Jurafsky and Martin, 2019) is the task of dividing text into individual words, numbers and punctuation or, as the name implies and more often used, tokens. While it may seem like a trivial task, the existence of special cases make this process more challenging than it initially appears. As an example, URL's, dates and prices should be kept as one single token, being the system function to identify these specific cases, dealing with them appropriately. While tokenizing, we can also improve the text by uppercasing (or down-casing) all characters and normalize the text by transforming abbreviations and numbers into their correspondent written words(Jurafsky and Martin, 2019). The following example shows how phrase (1) would be transformed in the sequence of tokens in (2):

An apple costs 1.05€. The Mr. wants 2 apples.

[this][apple][costs][one point zero five][euros][.][the][mister][wants][two][apples][.]

Sentence Segmentation

To understand the structure of a text, it is essential to identify where each sentence starts and ends. While usually a sentence ending is identified by punctuation mark, there are special cases that must be accounted for like abbreviations (for example, "Mr." and "Miss."). The process of sentence segmentation (Jurafsky and Martin, 2019) does just that, separating all the sentences in the given text as we can see in the following example.

An apple costs 1.05€. The Mr. wants 2 apples.

[An apple costs 1.05€.][The Mr. wants 2 apples.]

Lemmatization and Stemming

Lemmatization (Jurafsky and Martin, 2019) is the task of determining if two words have the same root. Essentially, verbs in different tenses, singular and plural words and different gender words are all considered to have the same root, which is how they will be processed. As a quick example, the words climbing and climbed would be processed as climb.

Stemming (Jurafsky and Martin, 2019) is a simplified version of Lemmatization, where instead of a deep analysis of each word, the suffixes are simply stripped from the words in order to achieve a good enough result.

Part-of-speech Tagging

As we mentioned previously, POS(Jurafsky and Martin, 2019), consists in giving each word specific lexical categories. While the most used categories are verb, noun, adjective and adverb, more can be used such as pronouns, prepositions, determinant, among many others. POS is extremely helpful in understanding the relations between a given word and its neighbors, giving the program a deeper knowledge about the overall context. An important English reference tag-set was introduced in (Marcus et al., 1993), including 45 tags used of POS.

2.2.3 Syntactic Phase

The syntactic phase, as the name suggests, deals with the syntax of the input text. The word syntax comes from the Greek *syntaxis*, meaning "setting out together or arrangement" (Jurafsky and Martin, 2019), dealing with how words are combined to form sentences, making some sentences valid and others invalid (Richards and Schmidt, 2013). During this phase, syntactic ambiguity can occur, where the true meaning of the sentence is not clear

due to various possible interpretations. As an example, in the phrase *I saw Carlos with the binoculars*, it is not clear if Carlos was carrying binoculars or if he was seen through the binoculars.

Dependency parsing is done during this phase, resulting in a parsing tree, which is helpful in minimizing syntax ambiguity as well as grammar errors. We mostly use this type of parsing in dependency grammars, which are important for speech and language processing systems, although other types of parsing such as constituency parsing exist and are used when in the presence of different type of grammars (Jurafsky and Martin, 2019).

Dependency Parsing

Dependency parsing (Jurafsky and Martin, 2019) is used in order to establish relationships between certain "head" words and "dependent" words. To illustrate this relationship, labelled and directed arcs are usually used.

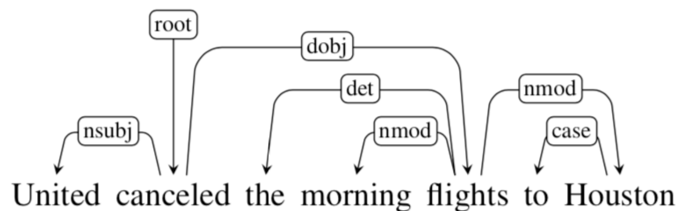


Figure 2.8: Dependency Parsing graph example(Jurafsky and Martin, 2019).

Each arc label describes the dependency that the arc is representing, initially defined and further expanded after as we can observe in figure 2.8, an example taken from (Jurafsky and Martin, 2019), with the phrase *United canceled the morning flights to Houston*.

2.2.4 Semantic Phase

The Semantic Phase (Jurafsky and Martin, 2019; Sarkar, 2016), aims at finding a meaning behind the input words and how they fit in a given context. While the lexical and syntactic phase deal with the overall structure of the input text, semantic only cares about the true meaning behind our input. Similarly to the previous phases, ambiguity can also occur here when the given sentence is capable of having more than one meaning. The main NLP task of this phase is Named Entity Recognition (NER), which we will briefly describe in the next paragraph.

Named Entity Recognition

In NER (Jurafsky and Martin, 2019), the goal is to find useful information about named entities in our input, where a named entity is anything that can be referred to with a name, be it a person, a company, a brand, etc. It is also usual to consider dates, prices, expressions, among others to be entities as they can be referred to by a specific name.

In table 2.1 we can see a short list as an example of existing NER types, adapted from Spacy[1]documentation. The number of entities to be used depends on the model used.

[1]https://spacy.io/api/annotation#named-entities

Type	Description
PERSON	People, including fictional
NORP	Nationalities or religious or political groups
ORG	Companies, agencies, institutions, etc.
EVENT	Named hurricanes, battles, wars, sports events, etc.
WORK_OF_ART	Titles of books, songs, etc.
LANGUAGE	Any named language.
QUANTITY	Measurements, as of weight or distance.
MONEY	Monetary values, including unit.

Table 2.1: A List of generic NER types.

2.2.5 Discourse Phase

The Discourse phase (*AI - Natural Language Processing* n.d.; Thanaki, 2017), commonly known as discourse integration, deals with the connection between sentences and the sense of continuity in the ideas transmitted by each of them. In other words, the meaning of each sentence depends on the meaning of the one that precedes it and contributes to the meaning of the succeeding one. As such, this phase deals with the coherence of the overall input text, such as whole paragraphs or even the whole document. Coherence analysis can be local and global.

Coherence Relations

There is more than one method for discourse organization and coherence. The most used method is Rhetorical Structure theory, where relations are defined between spans of text, defining a nucleus, a central part to the writer, and a satellite, a less central part that whose meaning usually depends on the nucleus. These relations are represented graphically, in a tree format with an arrow from the satellite to the nucleus (Jurafsky and Martin, 2019).

The second most used method is Penn Discourse TreeBank, where labelling is lexically grounded, using discourse connectives such as *because, although, when, since* or *as a result.* There are four major semantic classes, *Temporal, Contingency, Comparison and Expansion*, with each of them having types and subtypes that express relations between sentences (Jurafsky and Martin, 2019).

Due to their differences, each of these methods have individual techniques for structure parsing, a task whose aim is automatically determine the coherence relations between sentences.

Entity-based coherence

A discourse can also be coherent by being about some entity, where the discussion revolves on that entity. There are two theories of this kind, Centering Theory and Entity Grid. Centering theory bases itself on the fact that at a given time the overall text will be centered (hence the name) around a specific entity. Entity Grid is a technique based around a two-dimensional array, where columns are entities, rows sentences and cells represent the possible appearance of an entity in a given sentence, with the values representing if the entity has appeared and what is its grammatical role (subject, object, neither or absent) (Jurafsky and Martin, 2019).

Global Coherence

While the previous techniques only deal with local coherence, Global Coherence analysis (Jurafsky and Martin, 2019) is also present, with different types of discourse such as argumentation and scientific (among others) having different types of global coherence techniques.

2.2.6 Pragmatic Phase

Pragmatics (Richards and Schmidt, 2013) is the field of study in how language is used when communicating, especially the relationship between sentences, context and situations. The pragmatic phase (Thanaki, 2017), or pragmatic analysis, is the last step of a complete NLP pipeline. This phase consists in gathering real world knowledge of what is the true, real world meaning of the text (sometimes using external information) and how it was communicated.

2.3 Speech Synthesis

Speech synthesis (Oord, Dieleman, et al., 2016; Richards and Schmidt, 2013) is the artificial generation of speech. A system used for this purpose has the denomination of "Speech Synthesizer" with the final task of reading written text aloud in the most natural way possible, ideally without the listener realizing the speech is being generated by a machine. In a very simplified manner, human speech production is done by using movements and vibrations of muscles in conjunction with air-flow, generating periodic and aperiodic components. These components are then filtered and their frequency characteristics are modulated, generating speech. TTS systems aims to reproduce this process by computers.

In 1779, Christian Gottlieb Kratzenstein built a vocal track model that could produce the five vowels. In the mid-1800s, Charles Wheatstone showed an improved version of Kratzenstein's model that could also produce the sound of almost all consonants and even some words. However, it was only in 1939 that the first electrical speech synthesizer, called "the Voder", was developed by Homer Dudley at Bell laboratories. The Voder had a extremely high complexity degree, being almost unusable. With this said, Dudley also released another device known by the name of "Vocoder", divided in two parts, the first to analyze an incoming speech signal and the second to synthesize the sound, with this system still being seen as the basis of most TTS systems today.

TTS systems can be viewed as a sequence-to-sequence (Sutskever et al., 2014), where inputs and outputs may differ in size, mapping problem where a sequence of discrete symbols (text) are converted into a real-valued time series (speech). These systems are usually composed by two distinct parts, also referred to as the frontend and backend of the system. The first part analyzes text by applying a variety of NLP techniques such as the ones we saw before, outputting a phoneme sequence with a variety of linguistics contexts. The second part generates the speech waveform by taking the phoneme sequence as input, usually using prosody prediction and waveform generation.

One of the main uses of TTS, when in conjunction with STT, are virtual assistants. In fact, these systems are already present in most of our day-to-day lives under the name of

Siri[2], from Apple (Capes et al., 2017), Cortana[3] from Microsoft, Alexa[4] from Amazon and Google Assistant[5] from Google.

2.3.1 Key Concepts

This section aims to expose some concepts that are used in the TTS field of study, being commonly referred to when describing these systems.

Phoneme - A phoneme is usually considered to be the smallest unit of speech sound in a language. It represents how vowels, consonants, and some aggregation of both, sound. Words are composed by more than one sound and, as such, they represent a group of phonemes working together. The English language has about 44 different sounds which translate to 44 phonemes, but this number changes between different languages, as phonemes are language-specific.

Grapheme - A grapheme is for writing, what a phoneme is for speech, or in other words, the smallest unit of the writing system. However, a grapheme does not always correspond to a single phoneme.

Speech Naturalness - Speech Naturalness is a degree of how natural a given speech is to the listener. As an example, in speech synthesis, a robotic speech is considered to have a low degree of naturalness as the listener can easily understand it is not a real human being talking but a machine.

Intelligibility - Speech Intelligibility represents how clear the audio is and how easy it is for the listener to easily understand what the transmitted message is.

Prosody - Prosody can be defined as the natural aspects that influence how speech sounds, such as stress, irony or even the emotional state of the speaker. The differences caused by these variables are usually reflected in a modification of pitch, loudness, timbre and length of sounds.

Hertz - Hz, is the derived unit in the International System of Units and represents the number of cycles per second. The human ear is able to hear frequencies between 20Hz and 20000Hz.

Fundamental Frequency - Fundamental Frequency, also referred to as "Fundamental Frequency (F0)" or "Pitch", is the frequency of the lowest fundamental frequency in a given voiced sound. In humans, it is represented by the vibrations on the vocal coords being measured in Hz. A typical adult male voice will have a F0 of 85Hz to 180Hz and a female between 165Hz and 255Hz.

Decibel - A decibel, or dB, is a measurement unit corresponding to one-tenth of a bel. It is used to express power or intensity on sounds, tension, current, etc.

2.3.2 Sound Waves and Signal Processing Simplified

Sound is created with vibrations, specifically by sending waves of energy. These vibrations travel through air, moving air particles and creating sound waves. These sound waves can be graphically represented by waveforms, a representation of a wave's displacement

[2] https://www.apple.com/siri/
[3] https://www.microsoft.com/en-us/cortana
[4] https://developer.amazon.com/en-US/alexa/alexa-voice-service
[5] https://assistant.google.com/

over time, where displacement measures how far air molecules move when vibrating, being proportional to how loud a given sound is. Displacement is commonly represented in Y-axis and time on the X-axis.

When using a waveform representation, two main measurements are used. *Amplitude* represents the maximum displacement of a given wave, while *Frequency* measures how many times a waveform repeats itself in a given amount of time, a relation commonly measured in Hertz, and associated to pitch. When a soundwave is sinusoid it is called "pure". Most sound waves, however, are quite complex and composed by additional *Harmonics* and *Overtones*. Harmonics are additional frequencies, always multiples of the fundamental frequency and with lower amplitude, where the second harmonic has two times the frequency of F0, the third harmonic has three times the frequency of F0, and it goes on indefinitely. Overtones have a higher frequency than F0 but are not a multiple of F0. The addition of Harmonics and Overtones to F0 creates the sound *Timbre*, allowing the ear to distinguish different sounds with the same F0.

Finally, *Period* is the time required for the wave to perform a complete cycle in a given point. Period is the inverse of frequency, as in equation 2.6, so a higher frequency corresponds to a lower period.

$$period = \frac{1}{frequency} \tag{2.6}$$

Signal processing is an engineering subfield aimed at studying, analyzing and modifying signals such as sound and images. More recently, due to technology developments, more and more digital methods are used for this end, culminating in the creation of a Signal Processing subfield known as digital signal processing. While analog signals are continuous in time, being represented by waves, digital signals are discrete and represented by samples. Figure 2.9 depicts a comparison between the analog, or continuous, signal and a digital or discrete one.

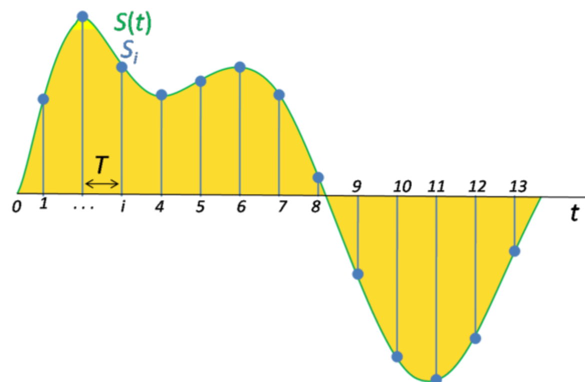


Figure 2.9: Continuous signal, represented by the green line, and a discrete represented by the various blue samples (*Sampling (signal processing)* n.d.).

The amount of samples per second on a given sound is specified by the sampling rate, in hertz or kilohertz , with a sample rate of 16 Khz or 16000 Hz translating into 16000 samples per second. Generally speaking the bigger the sampling rate, the crispier the audio is, however at a certain point the human ear can no longer distinguish the differences. With this said, the more common values are 16 Khz for speech signals and 44-48 Khz for music signals. When analyzing a given sound signal, it is common to use a representation called spectrogram, extremely used on modern TTS systems.

Generally speaking, a spectrogram contains three dimensions, time is represented on the X-axis, frequency on the Y-axis and the amplitude of the time-frequency bins, as seen in figure 2.10 on a color intensity (heat) map. A brighter color represents a higher volume whereas blacker colors represent lower volumes. It is important to note that in some representations the axis are switched (Frequency on the X-axis and Time on the Y-axis).

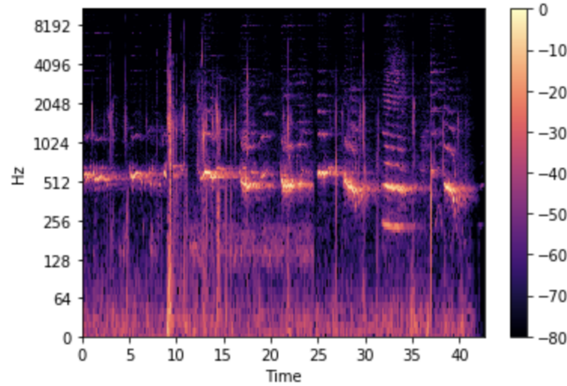


Figure 2.10: A representation of a Spectrogram in Mel Frequency Scale(Gartzman, 2019).

Mel-spectrograms (Umesh et al., 1999) use the Mel Scale, with the word "mel" coming from melody. This scale is based on the perception of pitch of the human ear, establishing a relation between real frequency and perceived frequency. As an example, a human would notice a bigger sound differences in frequencies between 100Hz and 200hz than between 11000Hz and 12000Hz, even if the total difference are the same 100Hz. As a general rule, the main point of reference between a mel-scale and a normal frequency is 1000 mels = 1000 Hz.

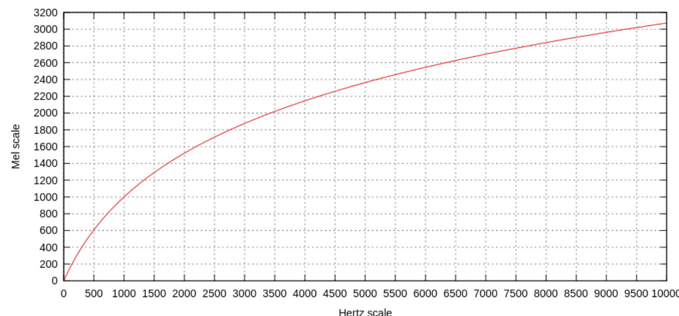


Figure 2.11: Mel to Hertz Scale(*Mel Scale* n.d.).

In figure 2.11 the full relation between mels and hertzs can be seen. It should be noted how it isn't a linear relation, as a consequence of human's decreased perception in sound differences as frequency goes up to considerable amounts. While there is no universal formula to represent this relation, most of the time equation 2.7 is used, where m represents mels and f represents the original frequency, in hertz.

$$m = 2595 \log_{10}(1 + \frac{f}{700}) \tag{2.7}$$

In order to go from a simple sound signal into a spectrogram, a technique called Fast Fourier Transform must be used in order to compute the Discrete Fourier Transform which outputs

the frequency contents of a signal. However, this technique just outputs the overall result instead of the time dependent one. To solve this, the Short-time Fourier transform (STFT) is instead used, computing the DFT on windows that slide across the full signal. A bigger window will increase the temporal resolution but decrease the frequency resolution, an effect known as time-frequency localization tradeoff (*Librosa - STFT* n.d.). Additionally, another variable usually referred to as hop length is considerably important, consisting in how many samples are skipped, or jumped, between each window analysis. Changing these two variables directly affects the final number of frames on the spectrogram. Choosing the right size of the window according to each problem is important, with bigger windows being more suited to music and shorter windows for speech analysis. The reason for this is that the amount of useful information across time is much higher on speech signals. Practical examples of how these variables interact with each other will be showed on chapter 4.

As a final concept, while dB are the overall used unit to measure sound level, Decibels relative to full scale (dBFS) is a unit also used, representing a measurement for amplitude levels in digital systems with a given maximum available peak. As an example, if a given system had a maximum peak at 30 dB, the equivalent value of dBFS at that point would be 0 and everything lower would have negative values.

2.3.3 Approaches

As we saw before, TTS systems are usually comprised by the frontend and backend, with each module being perfected as technology advances, always aiming at overcoming the main challenges of achieving high degrees of naturalness, intelligibility and prosody. However, there have been considerably bigger changes on the backend module, with two main approaches being developed. In this section, we will describe each one of these approaches and how they differ (Oord, Dieleman, et al., 2016).

Concatenative Approach

Concatenative Speech (Oord, Dieleman, et al., 2016) consists, as the name implies, in concatenating segments of previously recorded human speech. These segments, or utterances, are called units and are saved into databases. Most of the time, they consist of words, syllables, phonemes and other units of sound and speech in context positions. In order to achieve the best quality speech, one has to find the most suited unit length. However, there is a trade-off between shorter and longer units, where longer units are able to achieve a bigger degree of naturalness, due to having less concatenation points, and more control of coarticulation, but also require more recorded units in the database resulting in more memory used. On the other hand, shorter units require less memory but sample collecting and labeling procedures become more complex.

Overall, Concatenative methods are usually limited to one speaker and require more memory than other approaches although by having a large database, with a large number of recorded units with varied prosodic and spectral characteristics, it is easier to generate a more natural sounding speech.

Statistical Parametric and Deep Learning Synthesization

Parametric Speech Synthesization is considered to be the first truly data-driven approach, taking advantage of generative models in order to synthesize speech and training them on sets of data. Initially, statistical parametric approaches made use of Hidden Markov

models which would extract parameters and linguistic features from a speech representation in order to feed them to a vocoder, which would synthesize the speech waveform. When compared to concatenative methods, two main advantages arise such as the big reduction in the memory used, since a database with utterance is not needed in real-time and the capability to alter some parameters in the waveform, altering the final generated speech. However, as a downside, naturalness is often worse (Oord, Dieleman, et al., 2016).

Neural Network Speech Synthesis systems, as the name implies, use neural networks to achieve speech synthesization. While some authors also consider these systems as parametric approaches by using a generative approach to predict the vocoder parameters, it is mostly referred as a different deep learning approach. The main objective of these implementations is to synthesize speech while, at the same time, try to simplify the whole TTS pipelines by only using simple datasets with <text,audio> pairs without additional notations. The final aim is creating a completely end-to-end system instead of a multi-stage one, as we can see by the simplified evolution in figure 2.12.

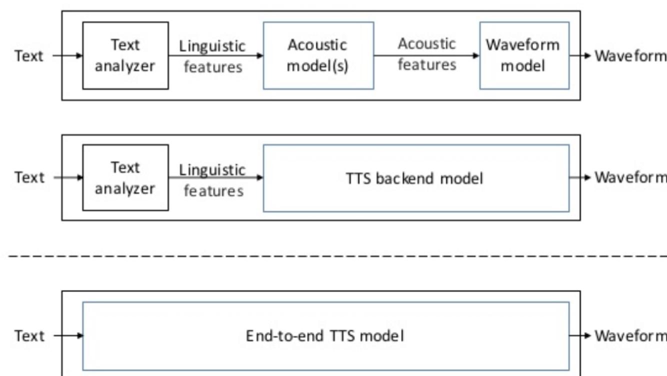


Figure 2.12: The overall TTS architecture evolution (X. Wang, 2019).

In this type of approach, it is usual for the frontend to output a Mel-spectrogram, a representation of the frequencies of a sound signal as it varies with time, used as an input for the backend vocoder. This spectrogram has the phonetic properties of the original input text encoded, used during the waveform synthesization.

As soon as the spectrogram is generated on the frontend it may be served as the input to the backend, taking into account that most current state-of-the-art vocoder implementations are expecting a spectrogram as the input. With this information, a speech waveform may be predicted and synthesized.

This approach is extremely dependent on the training data, as the quality of the original dataset may greatly increase or decrease the final quality of the synthesized speech.

Even with the quick improvements of neural speech synthesizers currently being made, this approach still suffers from flaws such as:

- **Robotic Tone -** The most common error and one of the main challenges of TTS systems is robotic tone where, as the name implies, the audio sounds similar to a robot, with a lack of naturalness. This error may be very simple to detect based on the sound characteristics and spectrogram if the robotic effect is extremely evident, or very hard if the effect is subtle.
- **Excessive silence -** Excessive silence, or stammering, occurs when the system repeats itself or makes short (yet excessive when in the middle of phrases) pauses

before continuing. This usually happens in certain words or phonemes. After the occurrence, the system may continue without further problems, although sometimes it may affect the generation, leading to more errors.

- **Random Artifacts -** Random artifacts are random and sudden sounds that are sometimes generated. Usually these have a very short duration although they are still noticeable when hearing the generated speech.
- **Critical Errors -** Critical errors occur when the synthesization goes wrong, completely ruining the generated speech. This can happen at any point of the sentence, with the generated speech having a much longer duration than the desired one and being composed by continuous and random sounds.
- **Background Noise -** Although not that common, it is possible for random noise to be generated alongside speech. The existence of noise in the training dataset can be the cause of this problem.
- **Pitch changes -** Similarly to random artifacts, the generated speech pitch is also subject to random sudden variations, although it is not that common for this error to appear.
- **Incorrect Prosody -** Prosody is one of the hardest components to synthesize correctly. Incorrect or stale prosody happens regularly in the generated speech. This error is complex to detect, being more easily corrected by improving the original dataset, similarly to background noise.
- **Long Sentences Struggles -** When the text to be synthesized is unusually long, the probability for the occurrence of the previous errors increases. Overall, the lengthier the text, the more errors appear. To a certain degree, this can be solved by simply dividing the phrase using punctuation.

These flaws are generally hard to predict, as a given trained model may be working perfectly for some phrases while struggling with others. As a final note, errors such as excessive silence and robotic tone are sometimes mentioned on the TTS community, but the above flaws were observed during our own experimentations. Some error samples can be heard on the following Google Drive link[6].

[6] https://drive.google.com/drive/folders/1BbkqyFl5B61Bi6L7LG57uLQVe48aeHQh?usp=sharing

This page is intentionally left blank.

Chapter 3

State of the Art

In the previous chapter we talked about how Text-to-Speech (TTS) synthesis has the final objective of generating an understandable speech waveform, given text as input. In this chapter, we will take a look at existing state-of-the-art solutions, how they are implemented, the methods they use to test these implementations and expose some public available datasets.

3.1 Existing implementations

Current implementations can be divided in three separated groups, Text-to-Spectrogram, VoCoders and End-to-End systems, according to their functions and architectures.

3.1.1 Text-to-spectrogram

Text to spectrogram implementations are considered the frontend of TTS systems. As we mentioned previously, this part is responsible for reading the given text data, preprocess it and map the given features and characteristics to a certain representations, which is usually a mel-spectrogram.

DeepVoice 3

Deepvoice is a TTS system developed by Baidu[1], a Chinese multinational technology company. Deepvoice had three iterations, Deepvoice 1 (Arik et al., 2017), Deepvoice 2 (Gibiansky et al., 2017) and finally Deepvoice 3 (Ping, Peng, Gibiansky, et al., 2017). The first iteration consisted of a single speaker model that could be trained without a pre-existing TTS system, which used a variant of Google's Wavenet (Oord, Dieleman, et al., 2016) for the vocoder. The system consisted of five main blocks:

- A grapheme-to-phoneme model based on an encoder-decoder architecture, that would use a multi-layer bidirectional with a Gated Recurrent Units (GRU) for the encoder and a deep unidirectional GRU for the decoder. This model had the final objective of converting written text into the corresponding phonemes.

[1] https://www.baidu.com/

- A segmentation model that would try to find the boundaries of each phoneme in the speech waveform part of the pair (text,speech). In other words, it would try to establish an alignment between the sequence of phonemes and a given utterance.
- A phoneme duration model for predicting the duration of each phoneme in a phoneme sequence, where the input consist in an hot-vector with information about the phoneme and the stress associated with that phoneme.
- A fundamental frequency model, that is actually part of the same neural network as the previous model. It is related with the prosody of the sentence by predicting wether a given phoneme is voiced and, in a positive case, also predicts the Fundamental Frequency (F0).
- An audio Synthesis Model, that essentially is a modification of Wavenet, for synthesizing audio by combining the output of the previous models.

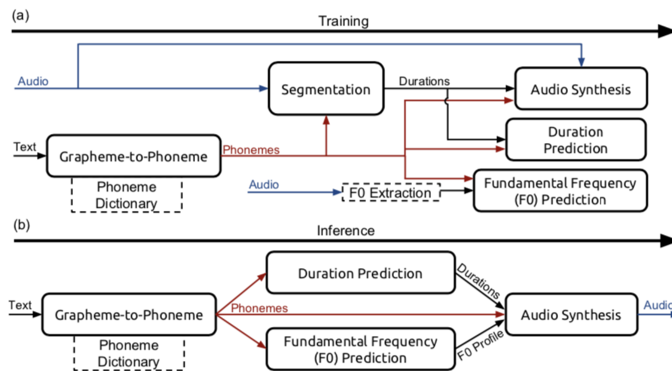


Figure 3.1: Original Deepvoice architecture (Arik et al., 2017).

Figure 3.1 represents the full overview of the original architecture of Deepvoice during training *(a)* and inference *(b)* as well as the links between the five models described. As said before, the duration prediction and fundamental frequency models are actually training in conjunction and form the same neural network.

The second iteration was based on a similar pipeline as Deepvoice 1 but improved the quality of each block and separated the previously connected phoneme duration and frequency models. In addition, Deepvoice 2 also introduced multi-speaker capabilities. To do this, the models were augmented with a low-dimensional speaker embedding vector per speaker. The parameters of each speaker are stored in a low-dimensional vector, which means that almost all weights are shared between them, avoiding additional unnecessary work. These speaker embeddings are included in multiple parts of each model as it wield better results than only including them on the initial input.

In figure 3.2 we can see this system in action, as each speaker embedding are included on various steps of segmentation *(a)*, duration *(b)* and frequency *(c)* models.

The changes made to the third iteration, Deepvoice3, were much bigger than between the previous two versions. This final iteration achieves a much more compact form by using a fully-convolutional attention-based character-to-spectrogram architecture, which also enables parallel computation allowing much faster training.

There are three main components, all of them fully-convolutional:

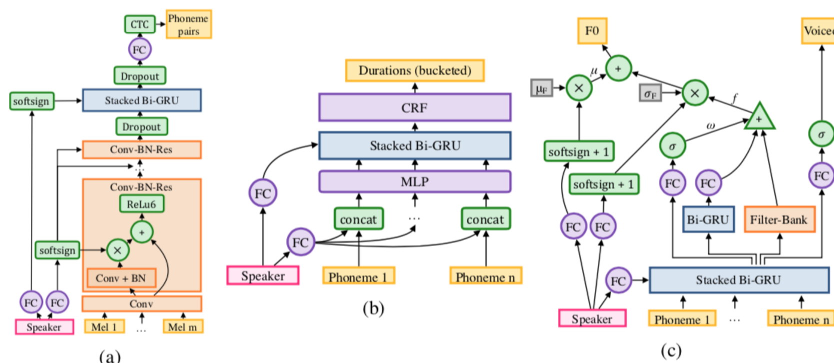


Figure 3.2: Multi-Speaker Deepvoice 2 Architecture (Gibiansky et al., 2017).

- An encoder used to convert textual features into an internal learned representation. The encoder starts with a embedding layer to convert the input into vectors. These vectors are then projected to a given dimensionality and go through convolution blocks to extract time-dependent text information. Finally, they are projected back to the original dimension in order to create attention vectors.

- A causal decoder, which takes the learned representation and decodes it into a low-dimensional audio representation such as a mel-spectrogram. By using auto-regression, it is able to predict future audio frames conditioned by past ones.

- A converter that predicts the final vocoders parameters from the decoder hidden states. These final parameters may change according to the choice of vocoder. The converter is non-causal and non-autoregressive being able to use future context from the decoder to predict its outputs. In order to prove that Deepvoice 3 was independent of the vocoder used, the authors adapted the outputs from the converter to three different vocoders (Griffin-lim(Griffin and Lim, 1984), WORLD(Morise et al., 2016) and Wavenet(Oord, Dieleman, et al., 2016)).

Tacotron2

Tacotron (Y. Wang, Skerry-Ryan, et al., 2017) was introduced in 2017 by Google as an end-to-end TTS system that could be trained in simple <text,audio> with minimal human annotation. Originally, Tacotron was based on a sequence-to-sequence model with attention, that would take characters as input and output the respective spectrogram frames. To do this, an encoder block with a complex module, named CBHG by the authors, was implemented. A CBHG module is composed by a 1-D convolution bank, a highway network (Srivastava et al., 2015) and a bidirectional GRU, to extract sequential features. The convolutional bank filters are used to extract local and contextual information. In order to preserve original time resolution, a stride of 1 is used. The highway-network, an architecture that introduced in (Srivastava et al., 2015) with the aim of simplifying the training of very deep neural networks, is used to extract high-level features, while the Bidirectional GRU is used to extract sequential features.

Just like Deepvoice 3, Tacotron also uses an encoder-decoder architecture where the encoder converts each character of the input text into an hot-vector embedded into a continuous vector.A set of non-linear transformations, which the authors call the pre-net, are then applied to each embedding with the outputs being fed to the CBHG to achieve the encoder

final representation used by the attention module. The decoder input is a concatenation of the context vector and the attention Recurrent Neural Networks (RNN) cell output, while the target is an 80-band mel-scale spectrogram as the authors considered a raw spectrogram to be highly redundant when trying to learn alignments between speech signal and text. Similarly to the encoder, the decoder also has a pre-net with dropout. Before the waveform synthesis, the output of the decoder goes through a post-processing net, consisting of a CBHG module, in order to get the vocoder parameters. For the waveform synthesization, the Griffin-Lim algorithm (Griffin and Lim, 1984) was used, however this was due to time constraints as the authors had the objective of making improvements at a later time. Figure 3.3 depicts the full architecture of the original Tacotron, from the character embeddings of the original input text to the final synthesized waveform outputted from the Griffin-Lim reconstruction.

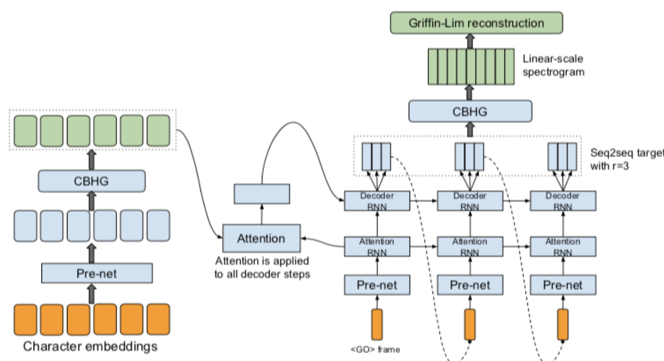


Figure 3.3: Original Tacotron architecture(Y. Wang, Skerry-Ryan, et al., 2017).

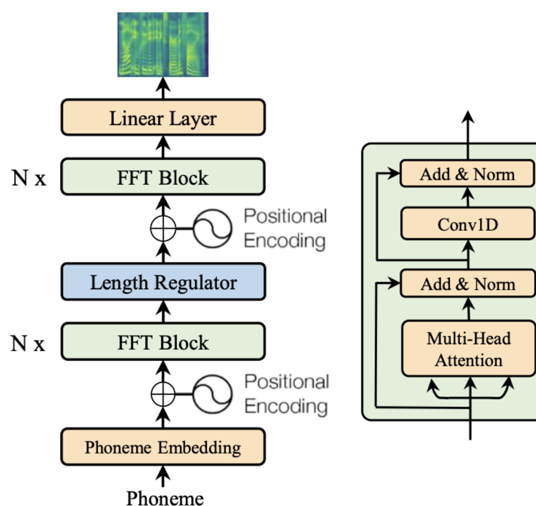
In 2018, Tacotron 2 (Shen et al., 2018) was developed, mainly replacing the Griffin-Lim algorithm of the original Tacotron for a modified version of the already existent vocoder Wavenet while simplifying the original building blocks. For example, instead of the CBHG modules and GRU recurrent layers in the encoder-decoder, Tacotron 2 uses vanilla Long Short-term memory (LSTM) and convolutional layers. At the time of writing, Tacotron 2 has seen improvements to better allow control over various multi-speaker parameters and generation, as we can see in the following work (Hsu et al., 2018; Y. Wang, Stanton, et al., 2018).

Fastspeech

Fastspeech (Ren et al., 2019) was published in 2019, with the main goal of increasing synthesization speed by making use of non-autoregressive approaches. Phonemes are extracted from the given input text in order to generate a log-mel spectrogram, using a feedforward network based on the self-attention Transformer (Vaswani et al., 2017), which was exposed on the previous chapter, and in 1D convolutions. This structure was named "Feed-Forward Transformer" on the original paper and can be seen in figure 3.4.

A length regulator is used to solve the problem of mismatching length between the phoneme and spectrogram sequence in the FFT, while also controlling the voice speed and part of the prosody.

Fastspeech was trained on the LjSpeech(see 3.3) and results almost match the ones achieved with auto-regressive models. In order words, there is a tradeoff between Speed and overall



.

Figure 3.4: Original FastSpeech architecture and FFT block (Ren et al., 2019)

voice quality. With this said, the quality was only slightly lower while the inference speed was several times quicker when compared to auto-regressive models.

3.1.2 Vocoders

Vocoders are the backend part of a TTS systems. They take the final representation of the frontend, usually a mel-spectrogram, and convert the information to waveform format in order to generate speech. In the following paragraphs, we will superficially describe the existing state-of-the-art implementations of vocoders.

Wavenet

Wavenet (Oord, Dieleman, et al., 2016) was introduced in 2016, implementing a fully probabilistic and autoregressive model by mainly using diluted convolutional networks.

According to the original Wavenet paper, a dilated convolution is a convolution where the filter is applied over an area larger than its length by skipping input values with a certain step.

Due to the nature of the input data, causal convolutions are used to avoid violations on the ordering when modeling, while allowing faster training than RNN's.

However, due to the causal nature, parallel inference is not possible although it can be used during training as a consequence of the known time-steps of the ground truth. During inference the predictions must be made sequentially, where each predicted sample is fed back into to the network in order to predict the next one. This means that Wavenet is extremely slow during inference, which presents a bottleneck for its use on a production environment.

Wavenets can generate audio with a set of required characteristics by conditioning the model on other input variables. Taking the example from the paper, in a multi-speaker setting the speaker can be chosen by feeding the speaker identity to the model as an extra

input.

To solve this problem, Parallel Wavenet was introduced in 2017 (Oord, Y. Li, et al., 2017), combining the capacity of parallel sampling in Inverse Autoregressive flows (Durk P Kingma, Salimans, et al., 2016) with the efficient training of Wavenet, creating, like it is said in the original paper, a new form of neural network distillation referred to as "Probability Density Distillation" where a already trained Wavenet model is used as a teacher for a feedforward IAF model.

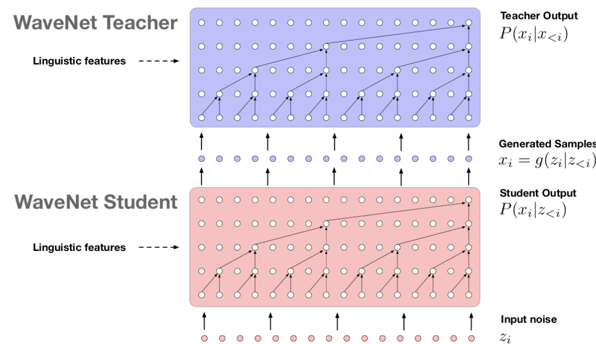


Figure 3.5: Overview of Probability Density Distillation (Oord, Y. Li, et al., 2017).

As we can see in figure 3.5 the main idea of the teacher-student framework is for the student to attempt to match the probability of its own samples under the distribution learned by the teacher.

Additional loss functions have been introduced as probability density distillation is not enough to assure the student network generates high quality audio. These are power loss, perceptual loss and contrastive loss.

Power loss ensures that the power in different frequency bands of the speech are on average used as much as in human speech, to avoid generating whispering. Perceptual Loss aims to penalize bad pronunciations. Finally, contrastive loss aims to minimize the Kullback-Leibler divergence, which essentially is a measure of how different one probability distribution is from another one, between the teacher and student when both are conditioned on the same information and maximizes it in different conditioning pairs.

The quality of speech achieved with this improvement had no significant difference, however the system was several orders of magnitude faster at inference than the original one. As such, Parallel Wavenet has been deployed in a production environment with success.

WaveGlow

Waveglow (Prenger et al., 2019) is an open-source Vocoder implementation proposed by Nvidia[2] in 2018 that uses knowledge from Wavenet and Glow (Durk P Kingma and Dhariwal, 2018) (Hence the name Waveglow), in order to achieve high-quality audio synthesis while rejecting the use of auto-regression and, as such, allowing parallel computation.

Glow is a work published by Nvidia, in 2018, that makes use of flow-based generative models to manipulate images. This same model type is used on Waveglow, resulting in a flow-based network that only uses a single cost function which results in simple and stable training.

[2]https://www.nvidia.com/en-eu/geforce/

While there are other solutions where auto-regression is not used such as Parallel Wavenet (Oord, Y. Li, et al., 2017) and Clarinet (Ping, Peng, and J. Chen, 2018), their implementations are far more complex to implement and train while achieving similar results to Waveglow. With this said, Waveglow only supports a single-speaker while the others support multi-speaker.

WaveRNN

WaveRNN (Kalchbrenner et al., 2018) was developed by Deepmind[3] in 2018 and, as the name implies, uses a single layer RNN's with a dual softmax (a mathematical exponential normalization function) layer and aims to reduce the number of parameters needed to produce speech,while maximizing their performance gains with the objective of requiring less computational power while sampling. The authors believe that large sparse WaveRNN networks tend to outperform small dense ones, where a large sparse network

In short, a sparse network is a network with fewer links, which means that weights are more relevant as each one has a bigger impact on the whole network. Dense networks are the opposite.

In order to prune the weights, a binary mask is used. At start time, the weight matrix is dense but every 500 steps the weights in each sparsified layer are sorted by their magnitude and the mask is updated by zeroing a set number of the weights with the smallest magnitude.

$$z = Z(1 - (1 - \frac{t - t_0}{S})^3) \tag{3.1}$$

Expression 3.1 represents the function used when pruning weights. Z is the target sparsity, t_0 is the starting pruning step in training and S is the total number of pruning steps. The set number of weights to be zeroed every 500 steps is a fraction of z.

To allow training in GPU's, a process based in sub-scaling is implemented. Sub-scaling allows the generation of multiple samples at once in a single batch,where a tensor of scale L is folded into B sub-tensors, each with a scale of $\frac{L}{B}$.

These sub-tensors are generated respecting their order, where each of them is conditioned by the previous one. While this could prove to be a handicap to parallel processing, only a small portion of the previous sub-tensor is needed and, as such, the generation of the next sub-tensor may start soon after the start of the generation of the previous one.

The small numbers of parameters and low requirements on memory band-with, allows implementations on low-power mobile platforms of real-time synthesis, being the first sequential neural model to achieve such feat.

3.1.3 End-to-End

End-to-End systems are capable of dealing with the whole process without the need of an independent backend or frontend.

Clarinet

[3] https://deepmind.com/

Clarinet (Ping, Peng, and J. Chen, 2018) was developed by Baidu as an improvement upon DeepVoice 3. It is the truly first end-to-end TTS system as it does not need a text-to-spectrogram or Vocoder to generate speech. It is a truly convolutional text-to-wave system. In this architecture, the spectrogram representation usually used to bridge the two main components is replaced by hidden states.

All the modules used in clarinet are convolutional, with the aim of avoiding RNN problems such as vanishing gradient while allowing fast training.

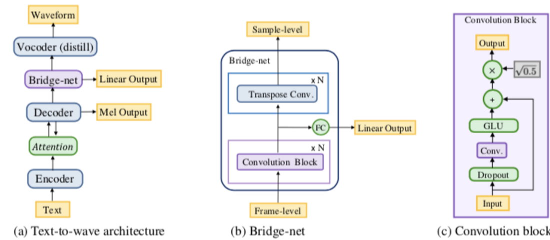


Figure 3.6: Overview of Clarinet Architecture (Ping, Peng, and J. Chen, 2018).

As we can see in figure 3.6, the architecture is composed by four main parts:

- An Encoder and a Decoder, that share their implementations with the ones in DeepVoice 3, with the decoder outputting a log-mel spectrogram.
- The Bridge-net, a convolutional block that processes the hidden representation from the decoder and predicts a log-linear spectrogram, while upsampling the hidden representation from a frame-level to a sample-level.
- The Vocoder takes the final hidden representation of the bridge-net to synthesize the waveform.

The waveform synthesization uses a teacher-student framework, similar to Parallel Wavenet, by using a Gaussian autoregressive WaveNet as the teacher-net and the Gaussian inverse autoregressive flow as the student-net. For loss computing, KL-divergence and power loss is used as only using KL-divergence resulted in whispering.

3.2 Evaluating Results

Evaluating the quality, mainly the naturalness and intelligibility, of speech is usually a very subjective task. As a consequence, the listening tests are usually regarded as the most reliable way of assessing audio quality.

The most common way of evaluation is Mean Opinion Score (MOS) where volunteers are usually asked to rate the audio in a scale of 1 to 5 in increments of 1 or 0.5 as we can see in table 3.5. These tests should also be done in a controlled environment alongside a large pool of subjects to guarantee the validity and accuracy of the test.

However, in recent years, most companies outsource the evaluation tests by crowdsourcing them, with the most used platform for this end being Amazon's Mechanical Turk (Kittur et al., 2008). Using platforms such as this, however, means that a controlled environment is no longer achievable as the hearing capabilities of the users, the quality of their surrounding

Written Score	Numerical Score
Excellent	5
Good	4
Fair	3
Poor	2
Bad	1

Table 3.1: The Mean Opinion Score ratings.

regarding sound isolation and the system outputting the sound can't be controlled. In 2011 Microsoft introduced CrowdMos (Ribeiro et al., 2011), an adaptation to MOS evaluation that takes into consideration the price of the crowd and the different variables that were introduced by the lack of control of the testing environment. In addition, CrowdMos also has available a toolbox to make it easier to integrate the tests with Mechanical Turk.

It is usual to compare the MOS results with a ground-truth value. A Ground-truth can be seen as the gold standard, the value that we should aim. In the case of Speech Synthesis, the MOS score of a true human speech could be considered the ground truth.

System	Type	MOS
DeepVoice 3 (WaveNet)	Text-to-Mel	3.78 ± 0.30
Tacotron 2	Text-to-Mel	4.526 ± 0.066
WaveGlow	VoCoder	3.961 ± 0.1343
WaveNet	VoCoder	4.41 ± 0.069
Clarinet	End-to-End	4.15 ± 0.25

Table 3.2: Overall state of the art systems comparison.

Table 3.2 represents a comparison of all the scores, with a 95% confidence interval, between all the systems and approaches from the previous sections.

It should be noted, however, that the MOS scores were taken from each individual paper, resulting in a different testing environment for each entry. As a consequence, this table only serves as a superficial comparison with no real scientific value.

3.2.1 Automatic Evaluation

Numerous systems have been developed with the aim of classifying a given audio sample in terms of overall quality. These systems are usually classified into intrusive and non-intrusive (Patton et al., 2016). Intrusive systems and measurements such as the Mel-cepstral distance (MCD) (Kubichek, 1993), PESQ (Rix et al., 2001) and POLQA (Beerends et al., 2013) have been developed. However, and by being intrusive, they are expecting a good quality sample alongside the original for comparison which is not available in TTS systems. While non-intrusive systems like ANIQUE (Kim, 2005), LCQA (Grancharov et al., 2006) and P.563 (Malfait et al., 2006) have been proposed, they are mainly aimed at evaluating telecommunication and telephony systems, not being ideal when focusing on certain speech characteristics. With this said, the MOS remains the best way to evaluate TTS implementations(Patton et al., 2016).

Since the MOS score relies heavily on manual and subjective human work, it was only natural for automatic systems, based on neural networks, to be developed. These systems mostly take into account a previous dataset of sounds with a given MOS as a label for each

sample. In the following paragraphs two of these systems are briefly described.

AutoMOS

AutoMOS (Patton et al., 2016) is a non-intrusive system, developed in 2016 by Google, for inferring the MOS score of a given audio in increments of 0.5, developed mainly using an architecture based on LSTMs whose input is the mel-scale spectrogram of a given sample. The original spectrogram is enhanced by adding information regarding it's velocity and acceleration (also known as delta and delta delta). Figure 3.7 depicts this implementation. Throughout the whole work, the authors give the following definition for synthesizer, directly citing them: *a synthesizer constitutes a snapshot of the evolving implementation of a unit selection synthesis algorithm and a continually growing corpus of recorded audio, combined with a specific set of synthesis/cost parameters.*

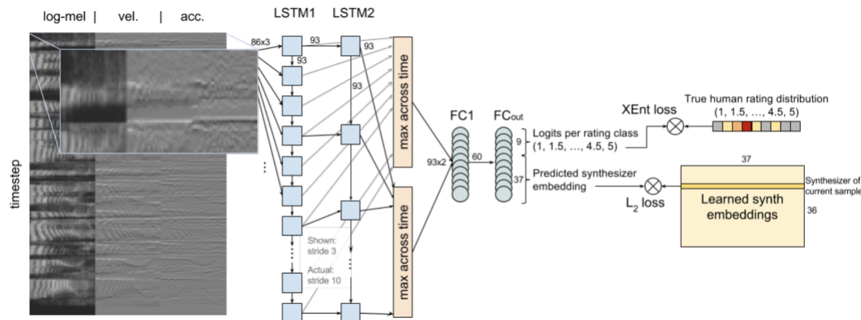


Figure 3.7: AutoMOS system architecture (Patton et al., 2016).

Regarding training and testing data, the authors use an internal dataset with samples and their respective naturalness MOS resulting from various year of developing and testing TTS systems. More specifically, the available data is composed by 168086 ratings across 47320 utterances from 36 different synthesizers, with each originating a number of samples between 64 and 4800.

To fine tune the hyperparams, Google's Cloud HyperTune [4] was used, with the best achieved results being displayed on table 3.3

In terms of results, AutoMOS tends to avoid the end of scale predictions (very high or very low MOS results), however this may be reflecting the distribution of the training data.

MOSnet

MOSnet (Lo et al., 2019) was presented in 2019 with the aim of presenting a neural network solution for evaluating voice conversion, which consists of trying to change a given speech signal of a source speaker to sound as if it was uttered by the target one while keeping the linguistic characteristics. While this implementation does differ from traditional TTS systems, it still uses MOS as an evaluation metric alongside the comparison between the original signal and the converted one. According to the paper, however, these two scores consistently achieve similar results, which allows the development of a system that only focus on the MOS.

[4] https://cloud.google.com/ai-platform/training/docs/using-hyperparameter-tuning

Description	Range Explored	Best Performer
Learning rate; decay / 1000 steps	0.0001 - 0.1; 0.9 - 1.0	0.057; 0.94
L1; L2 regularization	0.0 - 0.001	1.4e-5; 2.6e-5
Loss strategy	L2 | cross-entropy	cross-entropy
Synthesizer regression embedding dim	0 - 50	37
Timeseries type	log-mel | pooled conv1d	log-mel
Timeseries width (# mel bins, conv filters)	20 - 100	86
Timeseries 1-step derivatives	(none) | vel. | vel. + acc.	vel. + acc.
LSTM layer width; depth	20 - 100; 1 - 10	93; 2
LSTM timestep stride at non-0th layers	1 - 10	10
LSTM layers feeding hidden layer inputs	all | last	all
Post-LSTM hidden layer width; depth	20 - 200; 0 - 2	60; 1

Table 3.3: Overall best hyperparameters for AutoMOS

MOSNet bases itself on Convolutional Neural Networks (CNN), bidirectional long short-term memory (BLSTMs) and CNN-BLSTMs for extracting valuable features from an inputted spectrogram, interpreting the problem as a regression. The overall architecture of MOSnet has a considerable degree of complexity as shown in figure 3.8.

model	BLSTM	CNN	CNN-BLSTM
input layer	input (***N X 257 mag spectrogram***)		
conv. layer		$\left\{\begin{matrix} conv3-(channels)/1 \\ conv3-(channels)/1 \\ conv3-(channels)/3 \end{matrix}\right\} X4$ $channels$ = [16, 32, 64, 128]	
recurrent layer	BLSTM-128		BLSTM-128
FC layer	FC-64, ReLU, dropout	FC-64, ReLU, dropout	FC-128, ReLU, dropout
	FC-1 (***frame-wise scores***)		
output layer	average pool (***utterance score***)		

Figure 3.8: MOSnet system architecture (Lo et al., 2019).

In order to train, validate and test, a set of 20580 samples were used, divided into 13580, 3000 and 4000 samples respectively. Since the problem at hand consisted on speech, all the audios were downsampled to 16 kHZ. In terms of results, the best architecture was found to be CNN-BLSTM. At the date of the article writing, MOSNet was the first end-to-end speech objective assessment for voice conversion.

3.3 Datasets

Speech Datasets are not easy to generate as they required heavy human interaction. With that said, it is possible to find free datasets with really good quality, that were actually

used in some of the state-of-the-art approaches. In this section we will describe some of them.

LibriSpeech ASR Corpus

LibriSpeech (Panayotov et al., 2015)[5] is a collection of more than 1000 hours of 16 Khz read English speech. It was used in DeepVoice3 and was prepared by Vassil Panayotov with the assistance of Daniel Povey. This is one the biggest datasets, containing more than 60 GB of multi-speaker data across all the files.

LibriSpeech is structured in development, training and testing sets. They are structured by user ID, chapter ID and audio utterance ID's. The dataset consists of a group of folders whose name is the User ID. Inside the user ID folder, we can find folders with the chapters that were used, also referenced by an ID. Inside the chapter folder, there are the audio files, named with the structure "UserID-ChapterID-FileID.flac", and one transcript text file name "UserID-ChapterID.trans.txt" with the text from each of the audio files.

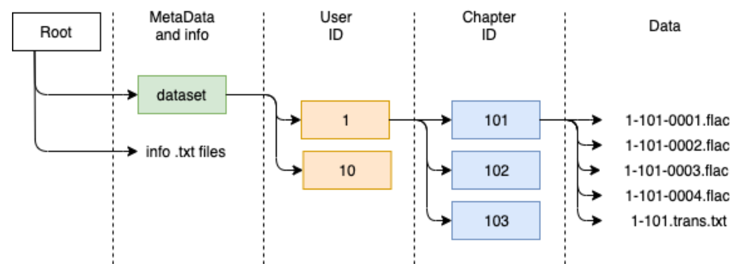


Figure 3.9: LibriSpeech Overall Structure.

In figure 3.9 we see the four audio files that were gathered by user 1 reading chapter 101 as well as the the full transcript from those passages.

LJ Speech Dataset

LJ Speech Dataset (Ito, 2017)[6] is a public domain speech dataset consisting of 13,100 short audio clips of a single speaker. The clips length varies from approximately 1 to 10 seconds and were automatically segmented according to silences of the recording, for a total of about 24 hours of speech that translates to 2.6 GB of data. It was used when training WaveGlow. This dataset structure is contained in a file called "metadata.csv". It then has a folder named "wavs" which contains all the audio files, each with a unique ID as the name. In table 3.4 we can see the overall structure of the dataset. Each entry in the column ID corresponds to a file with the same name in the folder "wavs". Then we have the text transcription of the audio on the column Transcription and in the column Normalized Transcription we have the same text but with numbers, ordinals, and monetary units expanded into full words.

CSTR VCTK Corpus

CSTR VCTK Corpus[7], originally developed for building HMM-based TTS systems (Yamagishi et al., 2009), is a multi-speaker dataset, containing data uttered by 109 native

[5]http://www.openslr.org/12
[6]https://keithito.com/LJ-Speech-Dataset/
[7]https://datashare.is.ed.ac.uk/handle/10283/2651

ID	Transcription	Normalized Transcription
LJ002-0298	in his evidence in 1814, said it was more	in his evidence in eighteen fourteen, said it was more
LJ003-0050	(1) those awaiting trial;	(one) those awaiting trial;

Table 3.4: LJ Speech Structure.

speakers of English, with various accents, where each speaker reads about 400 sentences. The data was recorded using the same device across all speakers and was further compressed and downsampled to a quality of 16 bits and 48khz. It has a size of about 10 GB and was also used in DeepVoice3, alongside the already mentioned LibriSpeech ASR Corpus.

Mozilla Common Voice

Mozzila Common Voice[8] is an initiative by Mozilla to build a high quality, free dataset of various voice speakers and languages.

It is currently under development and increasing in a daily basis. At the time of writing, it has 40 languages and 4257 recorded hours. The english dataset has 51072 different voices and 1488 hours of speech, which translates to about 38 GB of data.

3.4 Competition

There are currently available TTS solutions for developers, with the use of Application Program Interface (API)s. In general, these solutions have a good degree of quality but their extensive use in a production environment can be costly. They also support a large numbers of languages, voices and control over some variables but only a small part of them are synthesized using neural methods.

Google Cloud Text-to-Speech

Google Cloud Text-to-Speech [9] has more than 180 voices spanned across more than 30 languages generated using concatenative, parametric and neural methods (using Wavenet). As expected, the neural voices are considered premium and have a different pricing.

Additionally, it allows the modification of variables such as speed, pitch and volume and the introduction of detailed elements such as pauses. It is also possible to select profiles, developed to better suit certain target devices speakers such as wearables, phones, speakers, among others.

In order to connect to this solution, developers can use REST or gRPC api's protocols, accepting inputs with raw text or speech synthesis markup language (SSML). It is possible to convert the audio into base64 encoded audio and then decode it into mp3 format to save in a file.

Azure Speech Services

[8]https://voice.mozilla.org/en/datasets
[9]https://cloud.google.com/text-to-speech/

Azure Speech Service [10] is Microsoft solution regarding TTS. It has more than 80 voices (81 standard and 5 neural), a number that can be expanded by using voice customization with just a few minutes of training data, and 45 different languages. However, at the time of writing, voice customization using neural methods has limited access and requires an application in order to be enabled.

Variable modification such as rate, volume and pronunciation is also possible by using SSML. When using neural voices, the speaking style can be adjusted to better express cheerfulness, empathy, or sentiment.

IBM Watson

IBM Watson TTS[11] is available for 7 main languages, namely English, French, German, Italian, Japanese, Spanish and Brazilian Portuguese, using neural and standard generative methods. It supports various audio formats such as WAV, FLAC, MP3, among others.

Watson is available through an HTTP interface using REST API or WebSockets, allowing the modification of variables such as pitch, rate and timbre by using SSML.

Amazon Polly

Amazon Polly [12] is Amazon's TTS system. It allows two speaking styles, a Newscaster type, tailored to news narration and a more Conversational type for more general cases such as telephony applications. At the current time, it is available in 29 Languages with more than one voice per each language, including male and female variants.

Amazon Polly is available to developers with the use of a REST API and also supports variable modification such as loudness, pitch, speaking style and speech rate SSML.

Name	**Languages**	**Voices**	**Neural Voices**	**Integration**	**Feature Control**
Google Cloud TTS	30	187	95	REST; SDK	Yes
Azure Speech Services	45	81	5	REST; SDK	Yes
IBM Watson	7	34	14	REST, Websockets; SDK	Yes
Amazon Polly	29	61	13	REST; SDK	Yes

Table 3.5: Mean Opinion Score ratings.

3.5 Open-source implementations

A variety of TTS systems are available as open-source implementations, with some of these even providing pre-trained models. Each of these implementations bases itself on a given TTS architecture and are more often than not improved by a community. Since they follow an open-source philosophy, we did not see them as competition but as a possible opportunity, especially when most of the licenses used allow modification and commercial

[10] https://azure.microsoft.com/en-us/services/cognitive-services/text-to-speech/
[11] https://www.ibm.com/watson/services/text-to-speech/
[12] https://aws.amazon.com/polly/

usage. After going through a variety of repositories on Github[13], the ones with greater overall quality were the following:

- **Fatchord's WaveRNN**[14] - An implementation of WaveRNN using Pytorch[15], a python deep learning framework, and allowing Graphics Processing Unit (GPU) usage for training and inference. While WaveRNN was used for the backend, a Tacotron 1 implementation was used for the frontend. The author has available two pre-trained models, one for WaveRNN and one for Tacotron.
- **R9y9's Wavenet Vocoder**[16] - A Wavenet implementation using Pytorch. It is strictly the backend module and, in order to make inferences, the mel-spectrogram must be inputted instead of the original text. It also provides a pre-trained model.
- **Keithito's Tacotron**[17] - Tacotron implementation using Tensorflow[18], a machine learning framework. It also provides pre-trained models.
- **Nvidia's Tacotron 2**[19] - Nvidia provides their own Pytorch interpretation and implementation of the Tacotron 2 paper, using WaveGlow as the Vocoder. Models are also provided.
- **Nvidia's Waveglow**[20] - This repository is the official Nvidia implementation from (Prenger et al., 2019). It uses their own Tacotron 2 implementation, also open-source, for the frontend module. Models are available for use as a direct download or as a Pytorch import[21].
- **Mozilla TTS**[22] - This is Mozilla's official TTS repository, seen as a part of the Mozilla Common Voice project[23], and implements tacotron 1 and 2 using both Tensorflow and Pytorch. Regarding the vocoder, Griffin-lin is used.
- **ESPnet**[24] - ESPnet is an end-to-end processing toolkit mainly focused in TTS and Speech-to-Text (STT). It is considerably more complex than the previous presented projects, as it includes a variety of architectures and is maintained by a community of considerable size. Developed using Pytorch it includes a variety of models such as Tacotron, FastSpeech and Transformer as well as integrations with vocoders such as wavenet. Additionally, it also includes single and multi-speaker capabilities. It can be said that, at the time of writing, this is the definitive toolkit used across the open-source TTS community.

Across our experimentations, we used most of the described repositories to test various TTS implementations. However, ESPnet is considerably more advanced and active in all aspects when compared to the others. As a matter of fact, most of the authors of other repositories are now actively working and helping in ESPnet. These clear advantages regarding this toolkit led us to adopt it as our main tools for synthesizing speech.

[13] https://www.github.com
[14] https://github.com/fatchord/WaveRNN
[15] https://pytorch.org/
[16] https://github.com/r9y9/wavenet_vocoder
[17] https://github.com/keithito/tacotron
[18] https://www.tensorflow.org/
[19] https://github.com/NVIDIA/tacotron2
[20] https://github.com/NVIDIA/WaveGlow
[21] https://pytorch.org/hub/nvidia_deeplearningexamples_waveglow/
[22] https://github.com/mozilla/TTS
[23] https://voice.mozilla.org/en
[24] https://github.com/espnet/espnet

This page is intentionally left blank.

Chapter 4

Proposed Approach

As mentioned before, the initial goal of the internship was the development of an in-house Text-to-Speech (TTS) solution, for single and multi-speaker. However, after diving deeper into the field during the first semester, we concluded that training a new model with better performance than those already trained and publicly available would demand exceptionally powerful computational resources. Even if the hardware was easily available, it would take more than few days of non-stop processing to completely train the text-to-spectrogram and vocoder models. Furthermore, we concluded that the best way of significantly improving performance would be by retraining with a better quality voice dataset. Using Talkdesk's own database of call records was not an option, not only due to legal constraints but also due to the existent noise in those records, a consequence of all the compressions the sound went through before being stored. Our only option would be to build an internal dataset with professional speakers, resulting in extremely crisp and noise free sound. The time and resources needed to build this dataset, and then train the models without having the certainty of improvements, proved to be of extremely high risk.

After some thought, we decided to try to improve an existent solution while using the pre-trained models, by predicting when errors such as excessive silence, robotic tone, sudden artifacts and critical errors, such as the system blocking in a given word, would occur by analyzing the spectrogram that originates from the frontend and is the input of the backend. In case of detecting an error, a new, slightly modified text, would be generated to avoid the problematic words of the original text. In addition, we also agreed to add improvements such as background noise, simulating a working environment, and slight speech adaptations by detecting if, for example, the caller is in a hurry by speaking quickly. In the end, we believe these improvements would result in a more natural and personalized experience. However, aspects such as performance during run-time and overall quality would prove to be a considerable problem and the final goal was changed again.

With this final change, the goal was set to develop a evaluation system capable of quickly outputting a radar/spider chart[1] when the input was a group of synthesized audios from a given TTS system. This would prove to be extremely useful in quickly evaluating various implementations without loosing time in manually hearing samples.

The first section of this chapter will give an overview of the project requirements, where the main tasks will be listed. In the second chapter, the risks associated with the development requirements will be presented. The third section will show how the Talkdesk's team where the internship is taking place works, as well as some of the used tools. Finally, the fourth

[1]https://datavizcatalogue.com/methods/radar_chart.html

chapter will show the overall schedule of the internship through two gantt charts, one for the first semester and the other for the second semester.

4.1 Requirement Analysis

Software requirements (Wiegers and Beatty, 2013) are specifications of what should be implemented, describing how a given system or feature should work and behave. They not only express what should be implemented but also what may constrain this implementation. In this section, we will describe the main requirement for implementing the above solutions. Requirements are divided into four main types:

- **Business Requirements -** These requirements describe why the business, or organization, is implementing the system or feature and what benefits will originate from it.
- **User Requirements -** Description of tasks or goals an user should be able to perform and how it will achieve value from the system.
- **Functional Requirements -** Functional requirements describe what developers should be implement and how the system should work in a given environment, always fulfilling the business and user requirements.
- **Non-Functional Requirements -** These type of requirements often describe the environment where the system should operate as well as proprieties such as availability, performance, security, usability and even compliance, regulatory, and certification, among others.

4.1.1 Functional Requirements

As said before, functional requirements aim to describe what developers should implement, taking into consideration business and user requirements. They are usually described with an *ID*, *Name*, *Description*, *Priority* and *Dependency*. Each functional requirement may depend on other requirement, so it is only natural that the dependent one has a lower priority. In our project, we identified three different priorities, where 1 is high-priority and 3 is low-priority. The list of the functional requirements is the following:

ID: FR1
Name: Implement base TTS system.
Description: A base single-speaker TTS system should be implemented, making use of already existing solutions.
Priority: 1
Dependency: None

ID: FR2
Name: Adapt TTS system to extract spectrogram information.
Description: Adaptations should be made to the TTS solution in order to better extract information such as matrix and image representation about the generated

spectrogram during training and inference.
Priority: 1
Dependency: FR1

ID: FR3
Name: Build a robotic tone audio dataset.
Description: A dataset should be built containing various samples with different degrees of robotic speech, in order to have different classes.
Priority: 1
Dependency: FR1

ID: FR4
Name: Build excessive silence audio dataset.
Description: A dataset should be built, containing good audio and audio with excessive silence mid phrase.
Priority: 1
Dependency: FR1

ID: FR5
Name: Build critical errors audio dataset.
Description: A dataset should be built, containing audio that suffers from critical errors.
Priority: 1
Dependency: FR1

ID: FR6
Name: Build random artifacts audio dataset.
Description: A dataset should be built, containing audio that suffers random artifacts and noises mid phrase.
Priority: 1
Dependency: FR1

ID: FR7
Name: Build binary classifier to detect between robotic and non-robotic audio.
Description: An initial binary classifier should be built, with the aim of distinguish between robotic audio and clean audio. In this case the various degrees of robotic would not be considered.
Priority: 1
Dependency: FR3

ID: FR8
Name: Build 3-class robotic audio classifier.
Description: Improving the classifier developed in FR6, the aim is to develop a 3 class classifier that should take into consideration 3 different degrees of robotic audio, where the third class is clean audio.
Priority: 1
Dependency: FR7, FR1

ID: FR9
Name: Build 4-class robotic audio classifier.
Description: Improving the classifier developed in FR8, the aim is to develop a 4 class classifier that should take into consideration 4 different degrees of robotic audio, where the fourth class is clean audio.
Priority: 2
Dependency: FR8, FR1

ID: FR10
Name: Build 5-class robotic audio classifier.
Description: Improving the classifier developed in FR8, the aim is to develop a 5 class classifier that should take into consideration 5 different degrees of robotic audio, where the fifth class is clean audio.
Priority: 3
Dependency: FR9, FR1

ID: FR11
Name: Build binary classifier to detect extensive silence.
Description: A classifier should be developed to distinguish between normal audios and audios with an unnatural amount of silence mid-phrase.
Priority: 1
Dependency: FR4

ID: FR12
Name: Build binary classifier to detect phrases with critical errors.
Description: A classifier should be developed to distinguish between normal audios and audios that suffer from critical errors, where the speech eventually transforms into random gibberish.
Priority: 1
Dependency: FR5

ID: FR13
Name: Build binary classifier to detect artifacts.
Description: A classifier should be developed to distinguish between normal audios and audios that suffer from random artifacts, where there is suddenly a change on the overall pitch or speech.
Priority: 2
Dependency: FR6

ID: FR14
Name: Build mixed classifier, using different errors.
Description: A classifier should be developed to distinguish between different errors such as robotic tone, clean audio, excessive silence and critical errors.
Priority: 2
Dependency: FR7, FR8, FR11, FR12

ID: FR15
Name: Transform models outputs into radar chart.
Description: The various models output should be transformed into a radar chart, to quickly interpret the various results and evaluate a given system.
Priority: 2
Dependency: FR7, FR8, FR11, FR12, FR13

4.1.2 Non-functional Requirements

Non-functional requirements, as presented before, aim at describing constraints and metrics on how the overall systems should perform. The following were identified:

ID: NFR1
Name: Binary Models Accuracy
Description: The accuracy of binary models should be above 95%, since the two classes are very different and have unique aspects.

ID: NFR2
Name: Multi-class Models Accuracy
Description: The accuracy of models consisting of more than two classes should be above 80%.

ID: NFR3
Name: Programming Language.
Description: In order to maintain the integrity across the team's programming languages, Python or Java languages should be used.

ID: NFR4
Name: English Language.
Description: The system should be developed with a focus on the English language.

ID: NFR5
Name: Licenses.
Description: All the tools, data and libraries used must allow the use in a commercial environment and should not have associated fees.

4.2 Risk Analysis

Software development projects, similarly other kind of projects, have risks associated with them. These risks represent real problems that may arise during the development process, each with unique consequences, solutions and a certain occurrence probability. Identification and analysis of a project risks as soon as possible is of upmost importance in order to design mitigation strategies, reducing their overall impact(Vieira, 2018/2019).

Each risk has three main attributes, *Impact*, measuring how much this risk would impact the project or its threshold of success (TOS), *Probability*, measuring how probable is the occurrence of the risk be and finally *Timeframe*, measuring when would the risk possibly occur in the life of the project. Each of these attributes typically has three measurements:

- **Impact -** *Catastrophic* if project TOS becomes unreachable, *Critical* if TOS is only possible with great effort and *Marginal* if TOS is still possible without much effort.
- **Probability -** *High* if the probability is higher than 70%, *Medium* if between 40% and 70% and *Low* if it is lower than 40%.
- **Time Frame -** *Long* if longer than three months after the start of the project, *Medium* if between one and three months and *Short* if only a few weeks after the start.

Figure In this section we aim to expose the identified risks, classify them according tho the previous attributes and suggest a possible mitigation strategy in the case of occurrence.

4.2.1 Identified Risks

For this project, the following risks were identified:

Name: Technology Constraints.
Description: The intern may not have experience with the techniques and technology used for developing the solution.
Probability: High
Impact: Marginal
Timeframe : Short
Mitigation Strategy: In order to overcome this risk, the intern should study the

used technologies, do tutorials and ask his colleagues for help if needed.

Name: Hardware Constraints.
Description: The provided hardware by Talkdesk may struggle or be unable to process data(in the case of Graphics Processing Unit (GPU) computing) during the project, especially when training models and loading big data files.
Probability: High
Impact: Critical
Timeframe : Medium
Mitigation Strategy: In order to overcome this risk, a remote server with improved hardware can be used.

Name: Remote Hardware performance Constraints.
Description: If a remote server is being used, the provided remote hardware may still prove to be insufficient.
Probability: Low
Impact: Catastrophic
Timeframe : Medium
Mitigation Strategy: To overcome this risk, the team would need to ask for more financial resources to be allocated to the project.

Name: Remote hardware usage limit.
Description: The remote servers used may have a given threshold for GPU usage, only allowing training using CPU for a given time once this threshold is passed.
Probability: Low
Impact: Marginal
Timeframe : Medium
Mitigation Strategy: This risks may be mitigated by simply estimating the time needed for development and experimentation based on CPU training times, which is the worst case scenario.

Name: High Dataset creation complexity.
Description: The creation of the various dataset may prove to be harder than initially thought, since the errors must be generated artificially to generate a high enough number.
Probability: Medium
Impact: Critical
Timeframe : Short
Mitigation Strategy: The development of the datasets should be phased. For example, initially only develop the robotic dataset and models and, only once this section of the project is completed, start developing the next one. By doing this, it is

possible to make and guarantee progress in at least one classifier.

Name: Limited Dataset size.
Description: The amount of samples generated and used must achieve a balance between quality, generation time and training time. In the end, this could imply that our datasets are relatively small for the task at hand, which would have a negative impact on the final performance.
Probability: High
Impact: Critical
Timeframe : Medium
Mitigation Strategy: Due to our time-frame, the mitigation strategy passes for carefully studying how the models are performing with the given datasets and, as a last resort, slowly increase their size.

Name: Time Limitation.
Description: Developing all the proposed datasets and classifiers on the given time-frame may prove hard to achieve, especially since the development relies on external training conditions.
Probability: Medium
Impact: Critical
Timeframe : Long
Mitigation Strategy: Ignore the requirements with lower priority and focus on errors that happen more regularly such as robotic tone and excessive silence.

Name: Low Performance.
Description: Achieved model performance may be lower than expected.
Probability: Medium
Impact: Critical
Timeframe : Long
Mitigation Strategy: Improve the dataset, and search for possible flaws in the program and training methods.

4.3 Methodology

The company follows the software development processes introduced by the agile methodology Scrum (Rising and Janoff, 2000) and, as such, this is also the approach followed by the text-to-speech team.

Scrum is an incremental approach to building software with the aim of keeping high productivity and focus across the team during long periods of time while avoiding burnouts by finding the right working pace for each team. Following Scrum guidelines, Talkdesk is

mostly composed of small teams with less than 10 members. These teams are then grouped into larger clusters.

There are three distinct phases when approaching a new project:

- **Planning Phase** - Initially the team goes through a planning phase in order to define crucial aspects of the project such as the initial approach, state-of-the-art and competition analysis and architecture even though the latter can be changed at any time during development.
- **Development Phase** - After the initial phase, the development takes place, divided into smaller phases called sprints where each sprint can last from one to five weeks.
- **Final Phase** - This last phase is essentially the wrap-up and closure of the product development, taking place after the last sprint.

The current identified tasks are listed and organized by priority in the backlog, a special named list, at the start of each sprint. During the sprint, tasks are picked by the team members for development and new tasks should't be added to the current sprint backlog. Once the sprint ends, new tasks are be listed and the cycle restarts. It is expected that each sprint delivers a new part / improvement of the final product, being these the reason to calling Scrum an incremental approach.

In order to keep track of the team progress, short daily meetings are usually held where each members explains his current task, what he has done since the last meeting, current difficulties and what he will do next. Additionally, weekly internship meetings were held to track progress.

There are three main roles in a scrum methodology:

- **Product Owner** - Has the responsibility of understanding how the project should progress in order to achieve success, taking into consideration what the clients want.
- **Scrum Master** - The Scrum Master has the responsibility of delegating the work to the team while keeping track of the team productivity and burnout rate in order find the right working pace. A close relationship should be held between the Scrum Master and the Product Owner to ensure the project is going the desired way. All the Scrum meetings should be leaded by the Scrum Master.
- **Development Team** - As the name implies, is the team responsible for developing the product, consisting of two members at the time of writing.

In order to keep track of the tasks, two tools named Jira[2] and another called Trello [3] are used across the company. With these tools, teams can easily implement and follow the scrum methodology, assign tasks to each member and visualize various charts to better understand how the workflow can be further optimized.

For code versioning, the company uses Github [4], following a branching model called Gitflow [5]. In Gitflow, each feature is developed on a independent branch, only being merged into

[2]https://www.atlassian.com/software/jira
[3]https://trello.com
[4]https://github.com/
[5]https://datasift.github.io/gitflow/IntroducingGitFlow.html

the development branch when the code is ready. When a new feature is going to be developed, it should be branched from the development branch.

When a certain release is ready, a new release branch should be created and deployed into a test environment. After heavily testing this release, a merge to master and deploy to production can be made.

In order to develop, any IDE could be used, however Google's Colab[6] was chosen to be used since it also included free usage of Google's Central Processing Unit (CPU) and GPU hardware, even if this last one was sometimes limited according to usage, which is extremely useful for quick testing and prototype development.

4.4 Planning

This section will show how the overall internship was planned during the first and second semester by using Gantt[7] charts. To design them, we used the free online tools provided in TeamGantt[8] website.

4.4.1 First Semester

During the first semester, the internship mainly consisted of gathering knowledge on Machine Learning, Natural Language Processing and Speech Synthesis. During the first three weeks, Talkdesk provided activities and challenges to provide a better integration of the interns in the company culture. The Gantt chart in figure 4.1 aims at giving an general overview of how the internship was structured on the first semester.

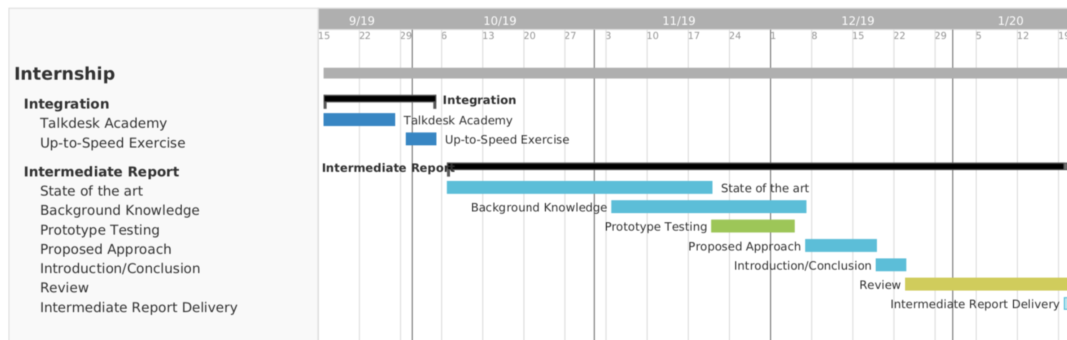


Figure 4.1: Gantt chart for the first semester.

Each section includes not only the writing of the report but also all the time spent gathering knowledge. We decided to start on the state of the art in order to see what the existing approaches were and what technology was being used, to research on them during the background knowledge phase. The review phase took longer than expected due to the changes on the internship scope, due to reasons explained in the previous sections.

[6]https://colab.research.google.com/
[7]https://www.gantt.com/
[8]https://www.teamgantt.com/

4.4.2 Second Semester

During the second semester, the internship consisted mainly in developing the presented solution, while trying to follow a total of 10 sprint phases. However, due to the nature of the project, this was not always possible. In figure 4.2, the gantt chart for the second semester can be seen.

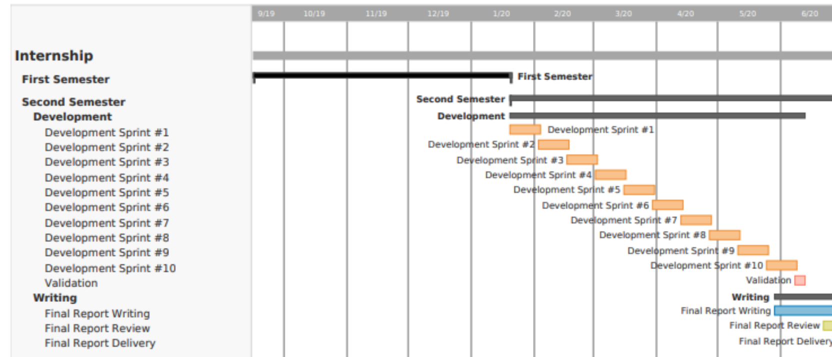


Figure 4.2: Gantt chart for the second semester.

4.4.3 Full Internship

This section aims at showing the conjunction between the first semester and second semester Gantt charts, seen on figure 4.3, to provide a complete overview of the full internship planning. As expected and exposed previously, the first semester was more focused on gathering knowledge, while the second semester had a greater focus on development

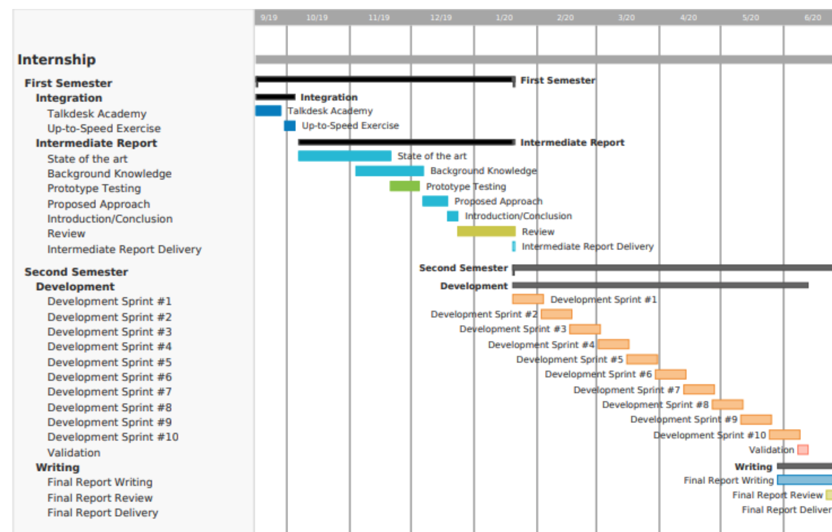


Figure 4.3: Gantt chart for the full internship.

This page is intentionally left blank.

Chapter 5

Development Approach

As initially explained in chapter 1, this work went through 3 distinct phases, where each one had a different end goal for the internship project. The first phase corresponded mostly to the first semester, where the goal was developing an in-house Text-to-Speech (TTS) solution. The knowledge gathered during the initial research, allowed the team to understand that achieving this goal with good results would not be feasible, not only due to the financial and time resources it would require but also due to the sheer complexity of the project. Additionally, the discovery of high quality, open-source implementations, meant we could just use this existing work as a starting point for our TTS system. However, these implementations had flaws on the synthesized speech such as robotic tone, excessive silence, random artifacts, among others.

This led the team to the second phase, the shortest one, where the new goal was developing a system capable of detecting synthesization errors and correct them during run-time in a production environment, avoiding outputting them to the listener (which, in a production environment, would be a client.). Essentially, this was an attempt at improving the existing implementations without having the need to re-train the already publicly available TTS models. Additionally, background effect such office noise would also be added to the synthesized sound, in order to improve realism to the listener. However, achieving the needed performance on detecting the error and solving during run-time was not realistic. Even if time was not a constraint, solving these kind of errors is a considerable challenge.

With this in mind, the third and last phase started, where the end goal was developing a system capable of evaluating a given TTS implementation by taking into consideration the more common errors. To do this, various classifier models would need to be built, trained and connected in a pipeline so that the output could be a graphical representation on the performance of the system we were evaluating. This was useful to quickly understand how good an open-source (or even closed-source) system was quickly, while also understanding its strengths and flaws.

In this chapter we describe the development process during the internship and the various phases, taking a look at the original requirements and how the final outcome compares to the initial planning. Since this work revolved mostly around experimentation, much of the information regarding how things were built on the third phase will be made on the next chapter, in conjunction with the obtained results. Additionally, a brief description of the work done on the second phase will be made, as well as a small overview on some of the tools that were used.

5.1 Developed Work

On the previous chapter, the functional and non-functional requirements, as well as the associated risks, were presented, with these being set before the actual development process began. By setting these goals at the start of a project, it is possible to have a general guideline of the expected work. However, it is not uncommon for the project outcome to fail meeting all the planned requirements, especially if any of the predicted risks came to happen. In table 5.1, the original goals can be seen as well as information regarding their completeness, where each color has the following meaning:

- **Green** - The requirement was completely fulfilled.
- **Yellow** - The requirement was partly fulfilled or was discarded during development.
- **Red** - The requirement was not fulfilled.

Requirement ID	Requirement Name	Status
FR1	Implement base TTS system	
FR2	Adapt TTS system to extract spectrogram information	
FR3	Build a robotic tone dataset	
FR4	Build extensive silence dataset	
FR5	Build critical errors dataset	
FR6	Build random artifacts dataset	
FR7	Build binary classifier to detect between robotic and non-robotic tone	
FR8	Build 3-class robotic tone classifier	
FR9	Build 4-class robotic tone classifier	
FR10	Build 5-class robotic tone classifier	
FR11	Build binary classifier to detect extensive silence	
FR12	Build binary classifier to detect phrases with critical errors	
FR13	Build binary classifier to detect artifacts	
FR14	Build mixed classifier, using different errors	
FR15	Transform models outputs into radar chart	

Table 5.1: Functional Requirements Status.

Generating random artifacts on purpose is considerably harder than generating other errors, which led us to give a lower priority to requirement FR6. Furthermore, since this error is not common, it was more logical to spend more time developing other datasets and models first. Critical Errors are easier to artificially generate, and the dataset development was initiated but went on hold when the time constraints became clearer. Since these datasets were not generated, the associated models were also not developed, leading to the not fulfilled status on FR5, FR12 and FR13. This fact was also the reason why the mixed model only was partially developed, simply including robotic and extensive silence errors.

The 5 class classifier for robotic tone requirement (FR10) was discarded after the development of the previous 4 class classifier, since the amount of data available was not enough for yielding minimum acceptable results, as will be explained on the next chapter. Finally, the radar chart, FR15, was not developed due to time constraints and lack of more error classifiers.

While most of the requirements were fulfilled during the third phase, where the final goal was more clear, FR1 and FR2 were actually developed between the start of the internship and the end of the second phase alongside the theoretical research. In order to implement FR1, various open-source systems were tested, each one with their own implementations. The goal was to search for various alternatives, comparing them in terms of how simple it was to make them work locally or on Google's Colab and how good their performance was, in both overall quality and inference speed.

As previously said in 3.5, ESPnet was chosen as the default system to use, since it allows the use of various algorithms and pre-trained models with only a few lines of code. Furthermore, it is being constantly updated by a community of users from very reputable companies. Of course that the constant updates also bring the occasional compatibility issues with previous versions and code, however the pros of being able to constantly work with the state of the art massively outweighs the cons in this case.

5.2 Interrupted Development

The adaptation of a TTS to extract spectrogram information (FR2) was developed during the short second phase and using this requirement's and FR1 output allowed us to better understand the problem at hand. Initially the main problems were identified and cataloged using an internal dialog dataset with 705 utterances spanned across 26 different dialogs. Each of these utterances was synthesized by the three ESPnet's models that were available at the time, Transformer, Fastspeech and Tacotron, for a total of 2115 audio files.

Each of these audios was then individually heard, in order to identify the existence of errors and classify them. By doing this, the aim was to start building a dataset for the future system to use. During the synthesization process, a record of various metrics regarding each utterance and each dialog was kept. In Table 5.2 and Table 5.3 an example can be seen for one utterance and one dialog, respectively. Additionally, for each utterance 3 images were saved, the internal spectrogram generated by the frontend to be fed to the vocoder, the final spectrogram from the generated audio and the final waveform.

The extracted metrics were used to check for patterns between audios with good quality and audios with errors, while the spectrograms allowed to understand if the problems were generated on the frontend or the backend of the TTS system. Most of the experiments showed the flaws already came from the frontend module. An initial architecture for this system was also designed and while some work was done regarding this implementation, it was interrupted.

5.3 Developed Classifier Models

Taking into account table 5.1, a total of 5 different classifier models were developed. The first three implementations focused on the problem of robotic tone, with each classifier having a different amount of classes. Naturally, the higher the number of classes, the more specialized it would be as it is able to distinguish more variable levels of robotic tone. The fourth implementation focus are prolonged silences in the middle of the phrases, or as we called it throughout this work, excessive silence. The final and fifth classifier mixes the previous implementations, trying to distinguish audios with a robotic tone, excessive silence and clear audio. These 5 implementations can be described as the following:

Utterance Metric Variable	Value
dialog_id	fastspeech_text
utterance_id	1
original_text	hello. Can you tell me, What type of movies do you like?
treated_text	HELLO. CAN YOU TELL ME, WHAT TYPE OF MOVIES DO YOU LIKE?
no_punctuation	HELLO CAN YOU TELL ME WHAT TYPE OF MOVIES DO YOU LIKE
total_words	12
total_letters	42
total_chars	56
mean_letters_by_word	3.5
audio_duration	3.99
mode	fastspeech
device	cuda
generation_time	0.52
frontend_time	0.09
frontend_spectrogram_time	0.37
vocoder_time	0.04

Table 5.2: Example of metrics taken for a single utterance.

Dialog Metric Variable	Value
id	fastspeech_text
total_dialogs	2
dialog_utterance_num	404
dialog_mean_utterance_num	202
dialog_total_words	4425
dialog_word_mean	10.95
dialog_total_chars	22579
dialog_mean_char	55.89
dialog_total_audio_time	1508.54
dialog_mean_audio_time	3.73
mode	fastspeech
device	cuda
dialog_total_generation_time	29.45
dialog_mean_generation_time	0.07
total_generation_time_frontend	0.25
mean_generation_time_frontend	0
total_generation_time_frontend_inference	13.19
mean_generation_time_frontend_inference	0.03
total_generation_time_vocoder	8.68
mean_generation_time_vocoder	0.02

Table 5.3: Example of metrics taken for a single dialog.

- **Binary classifier for robotic tone -** Binary classifier with the goal of predicting if a given audio file has a robotic tone or is considered good for a production environment.

- **3 Class classifier for robotic tone -** An improvement upon the previous model, which would classify the audio according to 3 classes. Class 1 has an extremely

robotic tone, class 2 has a slightly robotic tone and class 3 is good quality audio.

- **4 Class classifier for Robotic tone** - This model simply added one more class, splitting the previous model's third class. The fourth class was only comprised of audios from recorded human speech.
- **Binary classifier for excessive silence** - A binary classifier that would tell us if a given audio had excessive silence mid phrase. This would not take into account if the voice had robotic tone or not.
- **3 Class mixed classifier** - A mixed classifier, based on the 3 class classifier for Robotic tone, where the intermediate class was removed and replaced by a class characterized by excessive silences. This classifier not only detects robotic tones but also excessive silence.

The datasets used for these models were different according to the problem they were tackling and were mostly developed internally as it was needed. In chapter 6, a more in-depth view of each one of these classifiers will be made, as well all the development process and obtained results.

5.4 Main Frameworks and Libraries

In order to develop the datasets and error models, a variety of tools, including frameworks and libraries were used. This section an overview of these will be presented.

Sklearn

Scikit-learn[1], also known as Sklearn, is a free library for python, aiming at simplifying the process of implementing machine learning algorithms and neural networks. Methods such as classification, regression, clustering, etc. are provided in a very user-friendly format. In this work, the pure Sklearn library was used to implement k-fold validation across all classifiers in order to confirm the validity of our results and making sure the original division was not the reason of over or under-fitting.

Tensorflow

Tensorflow[2] is an open-source framework developed by Google, mainly used to develop machine learning, deep learning and mathematical projects. In this case, this project relies heavily on this framework, making use of the provided Functional Application Program Interface (API)[3]. Tensorflow is also heavily linked to Sklearn, making use of most aspects of this library and simplifying the interactions between the user and the framework, while allowing the use of GPU's processing power.

Librosa

Librosa[4] is an audio library with great capabilities aimed at manipulating and analyzing audios, be it speech, music or simply sounds. The developed work relies on Librosa

[1]https://scikit-learn.org/stable/
[2]https://www.tensorflow.org/
[3]https://www.tensorflow.org/guide/keras/functional
[4]https://librosa.org/librosa/

mainly for pre-processing tasks such as loading the dataset samples into time-series, making transformations into spectrograms and calculating the velocity and acceleration of a given audio.

Being more specific, the main used functions were the following:

- **Load**[5] - Function used for loading a given audio into a floating point time series. The default parameters were used except for the sampling rate (*sr* on the function) which was set to 16 Khz or 16000 Hz since most of the datasets audio samples were already using this rate as it is generally good enough for speech audios.

- **Melspectrogram**[6] - This function takes as input a given audio time-series and computes the associated mel-spectrogram. In terms of parameters, the used ones were the number of mels (*n_mels*) and the *n_fft*. The *n_fft* value will directly affect the hop length and window size, important variables seen in section 2.3.2, influencing not only the number of columns on the outputted spectrogram but also the amount of information it contains. More objectively, the window length will assume the same value as the *n_fft* variable and the hop length will be equal to $\frac{n_fft}{4}$, as long as the default librosa values for these variables are used.

  According to Librosa's documentation, the used *n_fft* value should be 2048 for music and 512 for speech processing, however tests were conducted using both values. As seen before, these variables not only affect the overall shape of the output but also other aspects such as the number of frames per second. The number of mels will essentially translate to how many rows the outputted log-mel time-series will have, directly relating to the number of features on our implementation.

  Using a more practical example in order to see how these variables interact with the output, lets take an audio sample with the total duration of 5 seconds. Since a 16 Khz sample rate is used, this audio will have $5 * 16000$, or 80000, total samples. If $n_fft = 2048$, then $hop_length = \frac{2048}{4}$ which is 512. This means the resulting spectrogram for this audio sample will have a total of $\frac{80000}{512}$, or 156.25, total frames. In order to achieve our frames per second, the total frames are divided by the total time in seconds which equals to 31.25 frames per second. If the same logic was applied to the *n_fft* of 512 the resulting frames per second would be 125 which contains considerably more information. Regarding the window size, if the value is equal to 2048, the time per window is equivalent to $\frac{2048}{16000}$ which is equal to 0.128 seconds, or 128 milliseconds. When analyzing speech, 128 ms is a considerably high value and phonemes and spectral variations may not be discriminated, reducing the overall amount of detailed information. On the other hand, if the value was 512, the time per window would be 32 milliseconds, a more suitable value to achieve a more detailed analysis. These simple calculation explains why Librosa advises different *n_fft* values for music or speech.

  Throughout our work, Librosa's default values for *hop_length* and *win_length* were used, which means that modifying the *n_fft* will be also affecting these variables.

- **Power_to_db**[7] - This function simply converts the given power spectrogram to use decibel units, essentially converting it to log scale.

- **Delta**[8] - Function used to compute the derivate along a given axis. In this case it was used to compute the delta, or velocity and delta delta, or acceleration. The outputs

[5] https://librosa.org/librosa/generated/librosa.core.load.html
[6] https://librosa.org/librosa/generated/librosa.feature.melspectrogram.html
[7] https://librosa.org/librosa/0.6.0/generated/librosa.core.power_to_db.html
[8] https://librosa.org/librosa/master/generated/librosa.feature.delta.html

were then concatenated to the original spectrogram increasing the total number of features by 3 times.

Pydub

The python pydub[9] library and packages are also aimed at extracting information from audio sources, much like Librosa. In our case, however, it was only used to develop a system that would help us assign stronger labels to the silence dataset, which we will describe in greater detail on the next chapter.

The main feature used was the **detect_silence**[10] function, which consists in an algorithm that detects if there is a segment with length bigger than a given value and loudness lower than a given threshold. In our specific case, the length must be bigger than 800 milliseconds and loudness must be less or equal than -40 Decibels relative to full scale (dBFS)

MatPlotLib and Seaborn

Matplotlib[11] and Seaborn[12] are two libraries aimed at simplifying the process of generating plots. They are widely used on the python ecosystem.

[9] https://github.com/jiaaro/pydub
[10] https://github.com/jiaaro/pydub/blob/master/pydub/silence.py
[11] https://matplotlib.org/
[12] https://seaborn.pydata.org/index.html

This page is intentionally left blank.

Chapter 6

Experimentation and Results

This chapter describes the development and experimentation process during the second part of the internship, as well as the obtained results. The first section will present a summary of the developed classifiers and main hyperparameters used, followed by a deeper dive into each one of them on the second, third and fourth section. The last section will give a final overview of all the classifiers.

6.1 Classifier Summary and Overall Structure

The main focus of this project went into the development of classifiers aimed at identifying the main Text-to-Speech (TTS) problems, including robotic tones on the synthesized audio and excessive silence in middle of phrases. While other errors exist and were identified, they were left as future work due to time limitations.

As seen in chapter 5 a total of 5 classifiers were developed, each built in a specific timeframe and inheriting most attributes from the previous ones. In order to simplify how the results are exposed throughout this chapter, each classifier will be associated to an acronym, which will be used as a prefix when explaining the various trained models of each classifier. The 5 classifiers, and associated prefixes, at the end of the internship were the following:

- **Binary Classifier for Robotic Tone (B-RT)**
- **3 Class Classifier for Robotic Tone (3MC-RT)**
- **4 Class Classifier for Robotic Tone (4MC-RT)**
- **Binary Classifier for Excessive Silence (B-S)**
- **3 Class Mixed Classifier (3MC-MIX)**

In table 6.1 the information relative to each model can be seen in a more compact format. Each of the models had the same set of hyperparameters, although with different values according to the given experimentation. Table 6.2 shows the final hyperparameters, a small description for each of them and if they were present on the original AutoMOS implementation (Patton et al., 2016).

Naturally, the total amount of hyperparameters increased progressively as the development advanced, starting only with the ones represented in AutoMOS, seen previously in table

ID	Classifier	Problem	Number of Classes
B-RT	Binary Classifier for Robotic Tone	Robotic Tone	2
3MC-RT	3 Class Classifier for Robotic Tone	Robotic Tone	3
4MC-RT	4 Class Classifier for Robotic Tone	Robotic Tone	4
B-S	Binary Classifier for Excessive Silence.	Excessive Silence	2
3MC-MIX	3 Class Mixed Classifier	Excessive Silence and Robotic Tone	3

Table 6.1: Developed classifiers overview.

3.3, and culminating with the ones seen on table 6.2. It is important to note that due to time-constraints, the attention given to each of them was not the same across our experimentations.

The size of the various sets was kept static through all the experimentations, with 80% for the training set, 10% for the validation set and the last 10% for the testing set. While all the results came from this training and testing techniques, cross-validation was also used to confirm the results. By doing this, we made sure the final accuracy values from both techniques were similar. 5 folds were used for classifiers B-RT, 3MC-RT, 4MC-RT and B-S and 10 folds for 3MC-MIX.

The hyperparameters regarding masking, truncation and deltas are directly related to the pre-processing of our data.

Setting the masking variable to *true* or *false*, toggled the use of a masking layer. Since dataset length was variable, padding was used to make sure all the inputs had the same length, a requirement of Tensorflow's LSTM implementation. This was especially useful when truncation was being used, since our dataset had very long and very short audios, as seen in figure 6.1, which would cause small input to be expanded with more than 2 or 3 times its size in zeros, until the length was the same as the longest dataset sample.

The truncation variable allowed to choose if the samples bigger than a specific length, computed by calculating the mean of the length of all audios plus the overall standard deviation, should be truncated in order to avoid outliers.

Finally, the delta variable allows the addition of the velocity and acceleration, also referred to as delta and delta delta, in each sample, effectively increasing the number of features by three times, a technique used on the original AutoMOS paper. In figure 6.2, a normal spectrogram is represented alongside it's individual delta and delta delta representation, as well as the spectrogram formed by concatenating the three representations. Note that the concatenated representation lacks the y-axis scale, since it would need to have 3 times the 0:8192 Hz scale of the previous plots as, in reality, it is composed by 3 different spectrograms, each with its own 0:8192 scale.

Hyperparameter	**Description**	**Present in AutoMOS**
Train Ratio	Percentage of the dataset to be used for training.	No
Test Ratio	Percentage of the dataset to be for Test.	No
Validation Ratio	Percentage of the dataset to be used for Validation.	No
Epochs	Max number of epochs to be ran.	No
LSTM Width	Number of Long Short-term memory (LSTM)s to be used. It should be between 1 and 2.	Yes
LSTM Depth	Number of units on the LSTMs.	Yes
Fully-Connected (FC) Width	Number of Fully Connected layers to be used. Should be between 1 and 2.	Yes
FC Depth	Number of units for the first fully connected layer, in case there are 2.	Yes
FC Output Width	Number of units of the last layer, also called output layer.	Yes
FC Activation Function	Activation function to be used on the output layer.	No
Loss Function	Loss Function to be used.	No
Global Metrics	The type of accuracy to be used.	No
Dataset Size	If all the dataset should be used or if it is limited. Useful for balancing all classes.	No
Cross Validation	If cross validation should be used instead of the fixed dataset split.	No
CV Fold Number	Number of folds when using cross validation.	No
Truncate	If truncation of the audio should be applied.	No
Deltas	If the delta and delta delta (velocity and acceleration) should be added to our input.	Yes
Masking	If the masking layer should be enabled.	No
Padding Type	How padding should be applied, at the start or at the end.	No
Truncate Value	Fixed value for when to truncate the audio.	No
Truncate Type	How truncate should be applied, at the start or at the end.	No
Optimizer	The optimizer to be used between Adagrad and Adam.	No
Learning Rate	The learning rate of our model.	Yes
Decay	The applied decay for each 1000 steps.	Yes
L1 Regularization	The value of the L1 regularization.	Yes
L2 Regularization	The value of the L2 regularization.	Yes
Number of Mels	The number of Mels to use.	Yes
Sample Rate	The sample rate to be used in each audio.	No
Pooling Size	The size of the pooling to be applied on the pooling layers.	No
Workers	The number of workers to be used.	No
Window Length	The size of the window used when generating the spectrogram from a given audio.	No

Table 6.2: Available hyperparameter, their description and presence on the original AutoMOS paper.

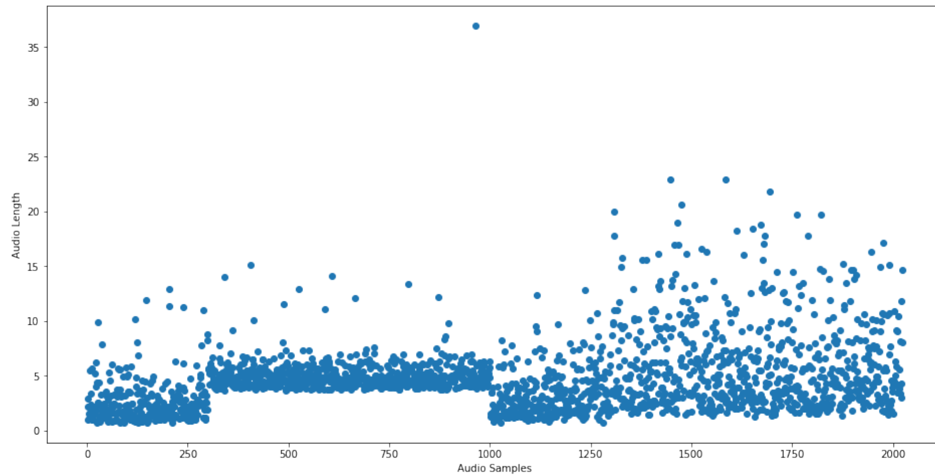


Figure 6.1: Scatter plot of audio lengths (in seconds) distribution.

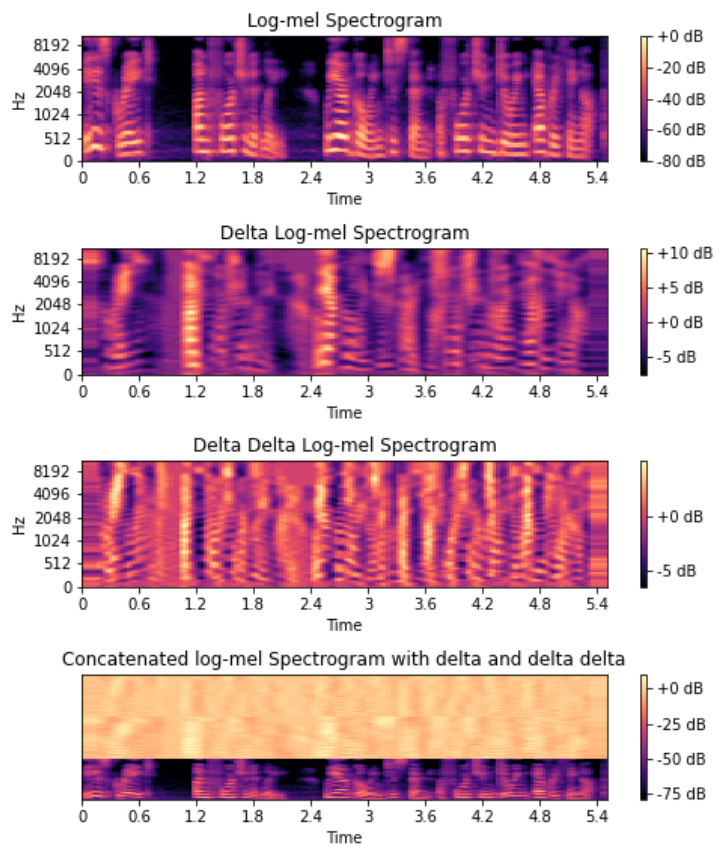


Figure 6.2: Spectrograms depicting the original, delta, delta delta and concatenated representations of a Tacotron audio.

6.2 Robotic Tone Classifiers

The first objective was the development of the robotic tone dataset and the associated robotic models. This section will dive into the first 3 developed classifiers, as well as the experiments done and the associated results. While initially only a binary classifier was

developed, it was further enhanced to allow classification of different degrees of robotic tones.

6.2.1 Binary Robotic Tone Classifier (B-RT)

Being the first classifier developed, it was naturally the one where most experimentation was done, which directly translates into a higher amount of time invested. From now on, it will be referred as B-RT. Furthermore, it was also by developing this classifier and its models that the knowledge regarding the used Tensorflow framework was gathered. As said previously, the aim was to develop a classifier capable of distinguishing between robotic tones, which consisted of various degrees of robotic and clear voice, that also had various degrees of how clear they were depending on the source.

The sources for our initial dataset were the following:

- Concatenative System - Using an old and outdated concatenative system, we were able to generate extremely robotic audios.
- Clone - This set of voices made use of an open-source project from github, named *Real-Time Voice Cloning* by *CorentinJ*[1] and the generated voices suffered from low quality and high noise.
- Fastspeech Model - Using the Fastspeech model provided by ESPnet.
- Tacotron Model - Tacotron Model provided by ESPnet.
- Transformer Model - Transformer Model provided by ESPnet.
- LibriSpeech Samples - Randomly selected samples from the LibriSpeech dataset, with more than one voice being present.

A characterization of the dataset used can be seen in table 6.3, where each class has roughly the same amount of audios, with 1000 audios for error class 0 and 1024 audios for the clear audio class 1. In total, the initial dataset had 456 MB of data, spanning across 224 minutes of audio. Samples for this dataset have been included on the following Google Drive link[2].

Audio Source	Number of Samples	Class	Categorical Numerical Class
Cloning	700	Robotic	0
Concatenative	150	Robotic	0
Fastspeech	150	Robotic	0
Transformer	150	Clear	1
Tacotron	150	Clear	1
LibriSpeech	724	Clear	1

Table 6.3: Robotic audio dataset used for the B-RT.

The development started by looking at Google's AutoMOS paper (Patton et al., 2016) as a starting point, whose architecture can be seen on figure 6.3. The starting point was developing the same architecture using Tensorflow, with slight modifications to the

[1]https://github.com/CorentinJ/Real-Time-Voice-Cloning
[2]https://drive.google.com/drive/folders/1Zr8nmQV61xafGK2eQPJkaEQCAwGxswUa?usp=sharing

output nodes, due to our desired binary output. Additionally, specific details of AutoMOS architecture were not explicit and, as a result, certain parts of the network were susceptible to our own interpretation of the paper. The developed Tensorflow network can be seen in figure 6.4

In terms of differences between the original network and our interpretation, the masking layer was added, as well as a pooling layer between both LSTMs, which is actually to implement the stride depicted on the AutoMOS version instead of actual pooling. The output layer was also changed, since our desired output was different.

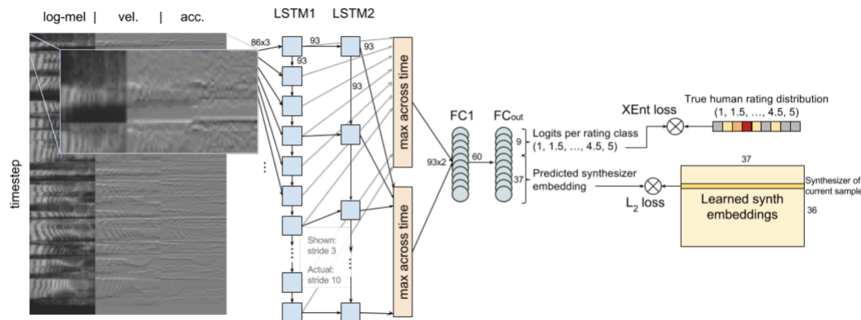


Figure 6.3: AutoMOS system architecture, already seen in 3.7 but duplicated here to improve reading flow.

Once the base architecture was complete, experimentation had to be done interactively in order to fine tune the hyperparameters. Taking into account the limited time, the approach taken was picking a first set of parameters to optimize and then, with the best combinations from that first set, optimize a second set of parameters, with the final results being considered our best architectures for the given problem. The distribution and tested values can be seen in table 6.4.

Variable	Values	Phase
LSTM Width	4 ; 16 ; 32 ; 64 ; 93	1
N_fft / Window Size	512 ; 2048	1
Masking	True ; False	1
Truncate	True ; False	1
Deltas	True ; False	1
Optimizer	Adagrad ; Adam	1
Mels	20 ; 40 ; 60 ; 76 ; 86 ; 96	2
Learning Rate	0.001 ; 0.01 ; 0.056 ; 0.1	2
LSTM Depth	1 ; 2	2
FC Width	0 ; 15 ; 32 ; 60	2
FC Depth	0 ; 1	1
Output Layer Width	1	1 ; 2

Table 6.4: Optimized hyperparameters values and corresponding phase.

It is important to note that the fully connected layer width (FC width) and depth variables were tested differently, even if they were also part of the second phase. While all combinations of the other variables were tested, the fully connected layer always had a fixed value for each value of LSTM Width. For example, the value of FC Width 15 was only used when the LSTM had 32 units since it was approximately half the units. The aim of this

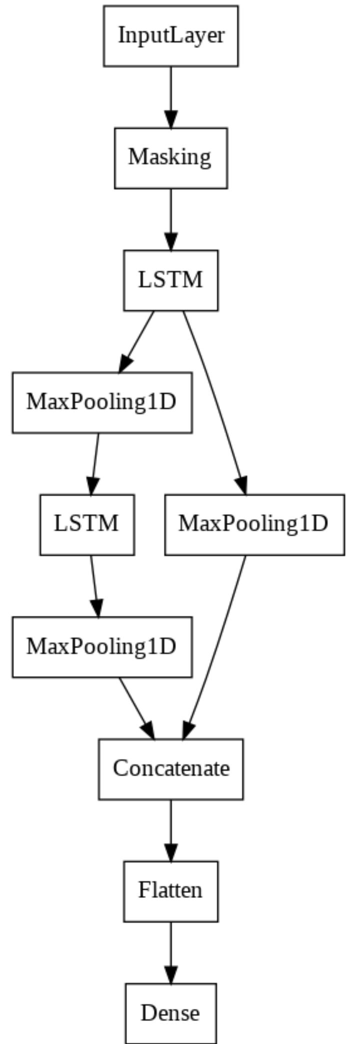


Figure 6.4: AutoMOS Network Tensorflow interpretation and implementation.

extra layer was creating what can be called a funnel, avoiding a big jump between a LSTM with various units to a very small output layer. Table 6.5 makes it easier to understand this relationship. Note that when the LSTM had 4 and 16 units, there was no intermediate fully-connected layer since the unit jump to the output layer was already relatively small. In the case of a LSTM with 93 units, the FC width was 60 since it was the value used on the AutoMOS implementation.

LSTM Width	Possible FC Width	Possible FC Depth
4	0	1
16	0	1
32	0; 15	1; 2
64	0; 32	1; 2
93	0; 60	1; 2

Table 6.5: Relationship between number of LSTM units and possible fully connected layers configuration.

Taking into consideration the chosen parameters combinations, more than 1000 model

experiments with different combinations were done. Once these were concluded, a large amount of model architectures were available. However, the model selection was not simply picking the overall best scoring ones. Instead, the best overall model for each amount of mels tested was chosen, in order to have models with various amounts of features (as the number of mels directly relates with number of features), increasing our overall diversity. In addition, and in the specific case of the 86 mels, an extra model using spectrogram deltas on the pre-process phase was also included since the original paper did use this technique. The result of this selection is exposed on table 6.6, with further classifier models only experimenting with these architectures.

	20 Mels	**40 Mels**	**60 Mels**	**76 Mels**	**86 Mels**	**86 Delta Mels**	**96 Mels**
Masking	False	False	False	False	False	False	False
Deltas	False	False	False	False	True	False	False
Truncation	True	True	True	True	True	True	True
Optimizer	Adagrad	Adagrad	Adagrad	Adagrad	Adagrad	Adagrad	Adagrad
LSTM Width	64	93	64	64	32	32	64
LSTM Depth	1	2	2	2	1	1	2
FC Width	32	0	0	0	0	0	32
FC Depth	1	0	0	0	0	0	1
Output Layer Width	1	1	1	1	1	1	1
Learning Rate	0.01	0.01	0.001	0.01	0.01	0.01	0.01
N_FFT and Window Size	2048	2048	512	512	2048	2048	512
Mels	20	40	60	76	86	86	96
Parameters (weights)	378177	129736	77249	81345	42817	42817	465473
Train Accuracy	93.1%	91.4%	89.8%	89.9%	91.0%	95.6%	88.2%
Validation Accuracy	91.3%	91.5%	93.0%	90.9%	88.6%	90.0%	90.8%
Test Accuracy	96.6%	96.0%	96.0%	95.5%	94.6%	94.6%	96.0%

Table 6.6: Best architectures for each number of mels tested. Note that the 60 mels model is the best performing one.

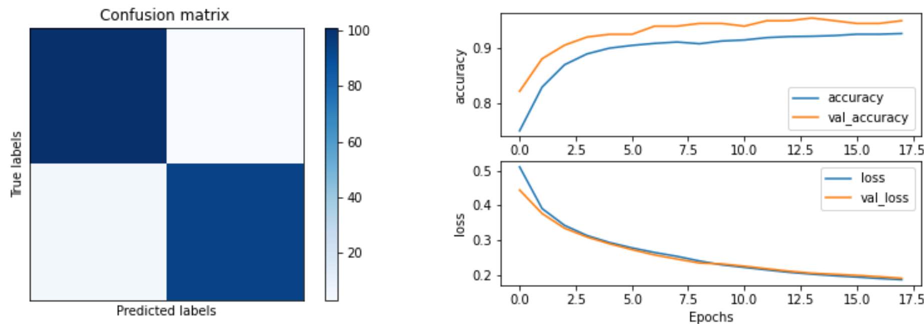
The number of parameters / weights depends on the used layers, expected outputs and input length. As an example, the 60 Mels model has 77249 parameters, corresponding to the sum of the weights in the first and second LSTM and the last output dense layer. If the input has length m, the number of LSTM parameters in Tensorflow is given by $4(mn+n^2+n)$, where n is the number of units. In the case of that specific model, the first LSTM had $4(64*60+64^2+64)$, or 32000, parameters and the second had $4(64*64+64^2+64)$ which is 33024 (n is 64, since the input here is the output of the previous LSTM). Finally, the last output layer comes after a flatten layer whose output is an array of size 12224. Since it has only one output node, the last layer parameters are equal to $12224 * 1 + 1 = 12225$. Summing all these values, $32000 + 32024 + 12225 = 77249$.

The overall accuracy of these models was above 90%, which is a very positive result even if

lower than the 95% threshold defined on the non-functional requirements. The confusion matrix for the test set of the best performing model, which is the model using 60 mels, can be seen in its numerical form in table 6.7, while image 6.5 shows it in a more graphical format alongside the plot depicting the evolution of accuracy and loss across training and validation epochs. Cross validation tests using k-fold validation, with 5 folds, were also ran separately to provide more certainty regarding the obtained results.

	Robotic	**Clear**
Robotic	101	3
Clear	5	94

Table 6.7: Numerical Confusion Matrix of the 60 mels robotic model.



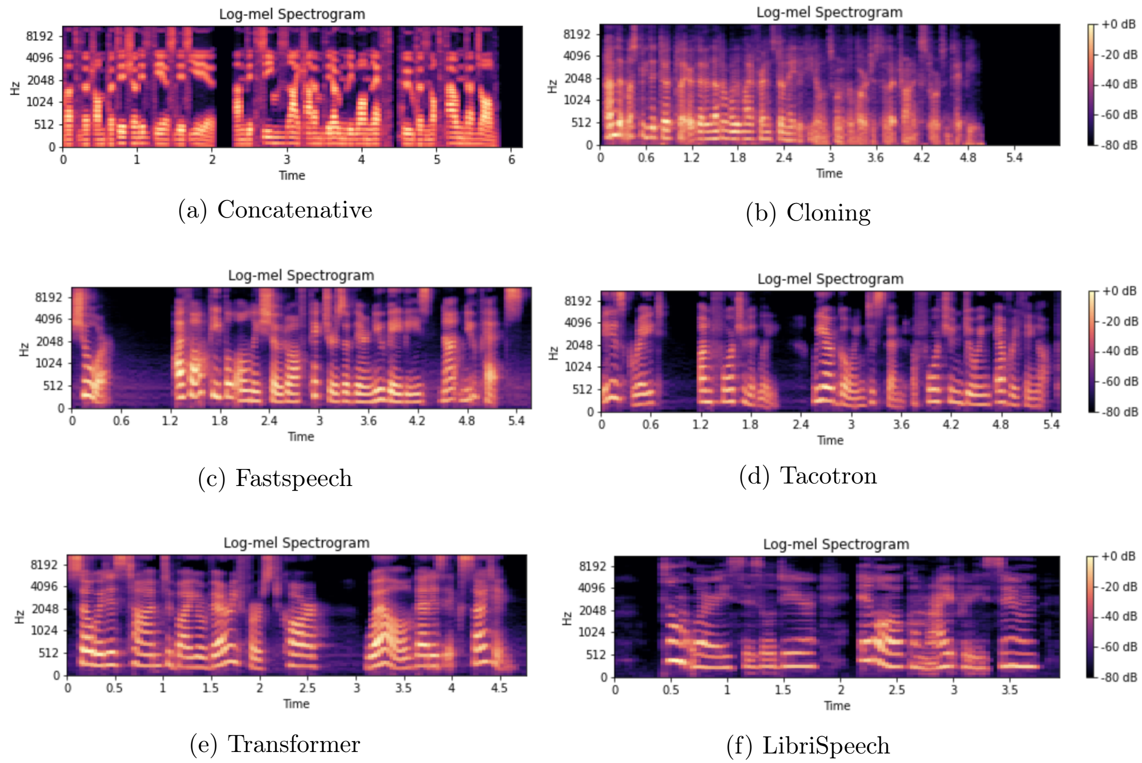
(a) Graphical Confusion Matrix of table 6.7.

(b) Accuracy and Loss across epochs for the training and validation sets.

Figure 6.5: Graphical Confusion matrix and evolution of accuracy and loss across epochs for the 60 Mel binary model.

In terms of architecture and parameters, improvements could probably be achieved by changing aspects such as decay, pooling size and L1 and L2 normalization. However, since the dataset has a very limited size, it may be the origin of this lack of accuracy.

Furthermore, the samples whose source is Fastspeech are considered as part of the robotic class but can sound very similar to samples from Tacotron and Transformer,which are classified as part of the clear audio class, since their speaker is similar as it was generated from the same dataset even if with different techniques. Figure 6.6 shows the 6 spectrograms, relative to each source of our dataset. Note that the presented spectrograms are not really comparable since they originated from audios with different contents in terms of speaker and phrase. However, by looking at each representation we can say that audios from concatenative and cloning sources are considerably different from the rest since their speakers have more unique characteristics. In the case of the concatenative samples *a*, the audio comes from a male voice, while in the cloning samples *b* the audio has a constant noise. Looking at the other four images, the similarities are evident, which emphasizes the thin separation between them. Actual experiments were done where Fastspeech samples were not included, which did yield inconclusive results, probably due to only 150 samples being from this source.



(a) Concatenative (b) Cloning

(c) Fastspeech (d) Tacotron

(e) Transformer (f) LibriSpeech

Figure 6.6: Spectrograms for the various 6 original sources.

6.2.2 3 Class Classifier for robotic tone (3MC-RT)

Once B-RT was concluded, the next step was improving that system in order to detect 3 different degrees of robotic voice tones, which can be referred to as extremely robotic audio, slightly robotic audio and clear audio. This classifier will be referred to as 3MC-RT.

The most evident change to the previous model was the activation function, as sigmoid is not well suited for multi-class problems. Hence, softmax was applied, which also changed how the loss and accuracy were computed on Tensorflow. The loss was manually changed from binary cross-entropy loss[3] to sparse categorical cross-entropy loss[4] and the accuracy is automatically changed by the framework according to the number of outputs and activation function used.

With the change from binary to multi-class, each audio had to be reclassified since we lacked multi-class labels. Ideally, the binary classification labels would have been obtained by merging the existing multi-class ones but, since at this time we did not have those, this method was not possible, which led to the complete reclassification. On a first approach, the audios were divided according to source. The resulting classes can be seen in table 6.8.

In order to keep the dataset balanced and due to the low amount of samples representing slightly robotic tones, or class 2, it was decided to limit the amount of audios of each class to 150. The 3 best models results can be seen in table 6.9.

The achieved scores were considerably positive taking into account how small the dataset was. In order to try to improve and give more confidence to the results, we decided to

[3]https://www.tensorflow.org/api_docs/python/tf/keras/losses/BinaryCrossentropy
[4]https://www.tensorflow.org/api_docs/python/tf/keras/losses/SparseCategoricalCrossentropy

Audio Source	**Number of Samples**	**Classs**	**Categorical Numerical Class**
Cloning	700	Robotic	1
Concatenative	150	Robotic	1
Fastspeech	150	Slightly Robotic	2
Transformer	150	Clear Audio	3
Tacotron	150	Clear Audio	3
LibriSpeech	724	Clear Audio	3

Table 6.8: First approach to 3MC-RT labels.

	3MC-RT Model 0.1	**3MC-RT Model 0.2**	**3MC-RT Model 0.3**
Samples per class	150	150	150
Masking	False	False	False
Deltas	False	False	False
Truncation	True	True	True
LSTM Width	64	93	64
LSTM Depth	1	2	2
FC Width	32	0	0
FC Depth	1	0	0
Output Layer Width	1	1	1
Learning Rate	0.01	0.01	0.01
N_FFT	2048	2048	512
Mels	20	40	76
Parameters (weights)	378243	172704	105795
Train Accuracy	80.8%	90.4%	95.2%
Validation Accuracy	86.8%	81.6%	79.7%
Test Accuracy	80.0%	80.0%	83.5%

Table 6.9: Initial 3 best models for 3MC-RT

strengthen our labels. Instead of simply classifying them according to source, the audios were listened and manual labels were given. It is important to note that in the case of the LibriSpeech and cloning sources, not all samples were listened, due to the large amount of files. During this process, we concluded that the voices whose source was cloning were not well suited for this dataset, as they actually only had a big amount of noise which didn't necessarily translate to being a robotic voice and would only cause confusion.

However, as our dataset had just got smaller, we had the need to increase it, especially for classes 1 and 2. To do this, more audios were synthesized. For class 2, we simply used the Fastspeech model as we had done previously. In case of class 1, we used the built-in mac voice of MacOS in order to have different robotic tones. Table 6.10 represents our final dataset for the 3MC-RT. Samples can be heard on the following Google Drive link[5].

Using our new dataset, we trained our defined set of 7 models, using the architectures from the previously seen table 6.6, in order to study the results. In addition, each of the architectures was further tested by setting a different limit to the number of samples used by each class. In other words, for each architecture a model was trained using 50, 250, 500 and all samples of each class.

[5]https://drive.google.com/drive/folders/1gF7cB9s2uYOmz0w8XBHRt9V6rPJfzAtn?usp=sharing

Class	Number of Samples	Categorical Numerical Class
Extremely Robotic	572	1
Slightly Robotic	538	2
Clear	731	3

Table 6.10: Final dataset for 3MC-RT.

In total, 28 models were trained for our 3MC-RT, with table 6.11 showing the five best models of these 28, when sorted by validation accuracy. Curiously, most of the models that used a dataset with only 50 samples of each class scored an high percentage in training, validation and testing accuracy, with the 4 best models being part of these.

	3MC-RT Model 1.1	**3MC-RT Model 1.2**	**3MC-RT Model 1.3**	**3MC-RT Model 1.4**	**3MC-RT Model 1.5**
Samples per class	50	50	50	50	All
Masking	False	False	False	False	False
Deltas	False	False	True	False	False
Truncation	True	True	True	True	True
LSTM Width	64	93	32	32	64
LSTM Depth	2	2	1	1	2
FC Width	0	0	0	0	0
FC Depth	0	0	0	0	0
Output Layer Width	3	3	3	3	3
Learning Rate	0.001	0.01	0.01	0.01	0.001
N_FFT	512	2048	2048	2048	512
Mels	60	40	86	86	60
Parameters (weights)	101699	172704	53955	31939	101699
Train Accuracy	81.0%	82.1%	75.3%	71.3%	81.1%
Validation Accuracy	96.7%	86.7%	85.3%	83.3%	83.0%
Test Accuracy	73.3%	80.0%	93.3%	66.7%	77.3%

Table 6.11: 5 best models for 3MC-RT, including models with 50 samples for each class.

However, having such a low amount of samples for each set reduces considerably these models credibility. As an example, our best model (3MC-RT Model 1.1) only had 4 samples for class 1, 6 samples for class 2 and 5 samples for class 3 in the test set, while our fifth best model (3MC-RT Model 1.5) had 63 for class 1, 50 for class 2 and 72 for class 3. The difference between class samples on each test set is due to how the dataset is randomly ordered, which directly impacts the class distribution on the last 10% of the dataset corresponding to the test set. With this in mind, it was decided that models trained with only 50 samples for each class should be discarded despite their good results. Our second iteration of 5 new best models can be seen in table 6.12. Note that the old model 3MC-RT 1.5 is now named 3MC-RT 2.1 since it was the best model not using a dataset sample size equal to 50.

Given theses results, there are 3 main aspects worth of our attention:

- The first aspect to note is that all 5 models use either the full dataset or 500 samples for representing each class. While not depicted in the table, the models that made

	3MC-RT Model 2.1	**3MC-RT Model 2.2**	**3MC-RT Model 2.3**	**3MC-RT Model 2.4**	**3MC-RT Model 2.5**
Samples per class	All	All	500	500	All
Masking	False	False	False	False	False
Deltas	False	False	False	False	False
Truncation	True	True	True	True	True
LSTM Width	64	93	64	32	64
LSTM Depth	2	2	2	1	2
FC Width	0	0	0	0	0
FC Depth	0	0	0	0	0
Output Layer Width	3	3	3	3	3
Learning Rate	0.001	0.01	0.001	0.01	0.01
N_FFT	512	2048	512	2048	512
Mels	60	40	60	86	76
Parameters (weights)	101699	172704	101699	31939	105795
Train Accuracy	81.7%	89.3%	85.7%	82.8%	81.7%
Validation Accuracy	83.0%	81.3%	79.0%	78.5%	78.2%
Test Accuracy	77.3%	83.8%	79.3%	83.3%	77.8%

Table 6.12: Final 5 best models for our 3MC-RT, after excluding the models where the dataset size was 50.

use of 250 samples had considerably worse results and, similarly to the ones with 50 samples, were discarded.

- The second aspect we should focus is that the architecture of all the best performing models for the different dataset sizes, including the previously discarded models of table 6.11, is the same, which could potentially imply that this is our ideal architecture for our final goal.

- Finally, the third aspect, is the number of parameters (or weights) on all our top ranking networks, where the maximum value is 172.704, with the rest being around, or below, 100.000. By comparing to our lowest scoring models (not depicted on any table), which have around 400.000 parameters, it is easy to understand we are actually taking benefits in using simpler neural networks. This actually makes sense, as our full dataset still is actually small for our task, and training a bigger number of parameters would make this more evident.

Diving deeper into the results and focusing on 3MC-RT model 2.1, which is our overall best for this classifier, table 6.13 shows us the numerical confusion matrix from the test set, while figure 6.7 shows the exact same matrix in a more graphical, but less detailed, representation. The overall architecture regarding the used layers is the same as the previous model, seen in 6.4.

By looking at these matrixes, it is evident that class 1 is being well classified and distinguished, which was expected due to being so unique when compared to the other two classes. However, it is also clear that the model is getting confused when classifying audios from classes 2 and 3. While this lack of accuracy represents a flaw, we were already predicting it. The problem here is that those classes are represented by very similar audios,

	Class 1	Class 2	Class 3
Class 1	59	0	1
Class 2	2	35	22
Class 3	2	15	49

Table 6.13: Numerical Confusion Matrix for 3MC-RT model 2.1.

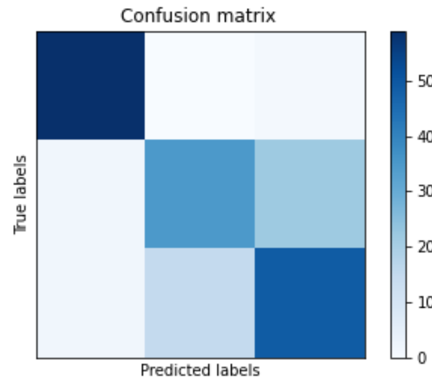


Figure 6.7: Graphical Confusion Matrix for 3MC-RT model 2.1.

which were manually labeled on the dataset based on the subjective opinion and hearing capabilities of the listener. In fact, we made an experience with various listeners which resulted in a single sample being classified differently, proving the degree of subjectiveness when doing this kind of labeling.

This degree of subjectiveness in conjunction to the low amount of samples currently on the dataset, made this outcome expected. In figure 6.8, the evolution of training and validation accuracy, as well as loss, of our model across the epochs can be seen, with the convergence happening before our defined max number of epochs. We strongly believe that increasing the amount of samples, and the strength of their respective labels, might allow us to improve this aspect, allowing the model to converge later and improve the results of our model, while keeping the current architecture.

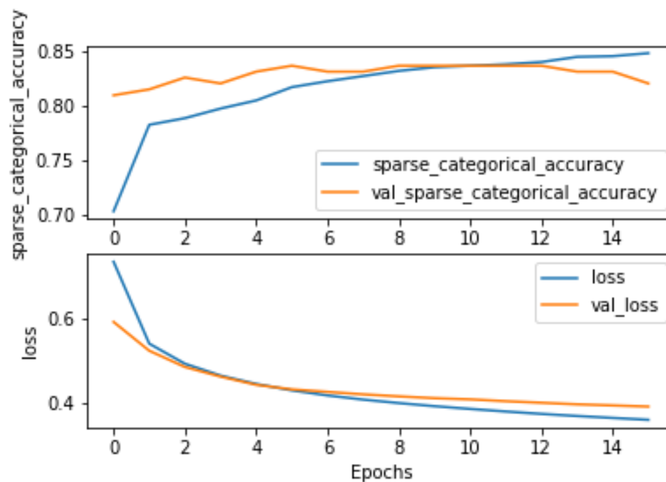


Figure 6.8: Accuracy and Loss for 3MC-RT model 2.1

On an additional note, while class 1 had 4 misclassified samples, we found out that they were actually well classified by the model and the problem was on the actual human inputted labeling, which shows the model actually correcting human error.

6.2.3 4 Class Classifier for Robotic Tone (4MC-RT)

The 4 class classifier for robotic tone, 4MC-RT for simplification, is simply an extension of the previous 3MC-RT. The aim is essentially the same, detecting various degrees of robotic voices. When compared to the previous classifier, the main difference is the existence of a fourth class, generated by simply splitting the previous class 3. Simply put, the audios previously classified as class 3 whose source was a TTS model kept its label and the ones where the origin was a dataset from a real human (LibriSpeech) were assigned the new label of class 4. The exact number of samples per class can be seen in table 6.14. Samples are available on the following Google Drive link[6].

Audio Source	Number of Samples	Categorical numerical class
Extremely Robotic	572	1
Slightly Robotic	538	2
Clear synthesized	397	3
Clear Human Voice	334	4

Table 6.14: Dataset used for representing 4 robotic classes.

With the knowledge acquired from the previous experience in building and training 3MC-RT, it was already known the models for this classifier would get confused, especially since the amount of samples did not increase despite one more class being added. With this said, however, the trained models did achieve surprising and curious results.

Table 6.15 represents the 3 best models for this classifier, with all of them having less than 200.000 parameters. Once again, this comes to emphasize that a simpler network works best with our current dataset.

By looking at the various accuracy scores, we can, curiously, note that they are actually not much different than the ones attained when classifying three classes. By adding one more class, we were expecting the confused behavior between class two and three to be extended to class four.

Taking a look at the confusion matrix of 4MC-RT model 1, depicted on figure 6.9 and table 6.16, it can actually be seen the model had no problems in differentiating class 4, which directly translates into the confusion existing between the samples coming from synthesized sources. For comparison, 4MC-RT model 2 confusion matrix was also included in figure 6.9, which has the same problem even if it is not as one-sided as 4MC-RT model 1. Once again, regarding the layer architecture, it remained the same as the previous classifiers, seen in 6.4.

While not being our original goal, by conducting experiment with four classes it allowed us to have a better understanding about which samples were causing confusion. While class 4 samples have great quality, they also have very unique traits since it comes from real audiobooks. The speech characteristics have much more emotion and prosody when compared to other classes, which could justify why it is being classified with so much accuracy despite having a smaller amount of audios. The results obtained training this

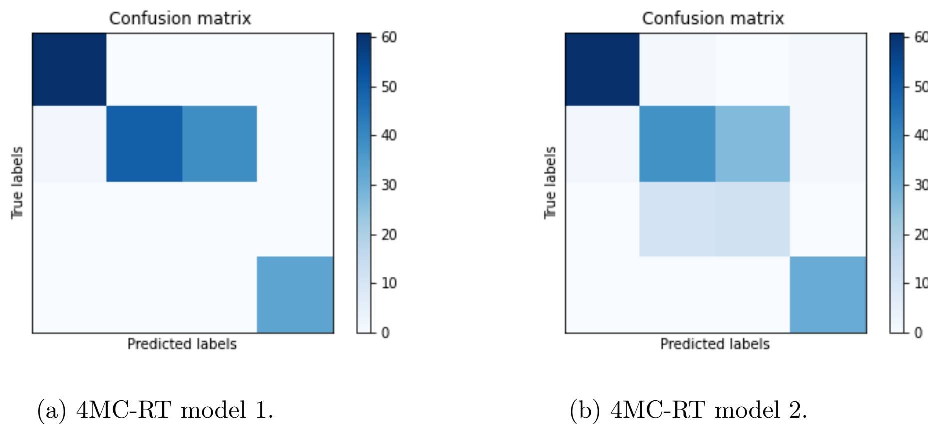
[6]https://drive.google.com/drive/folders/1TCMEYrHvjTVSL5kxPNb-4hsl6jAEd51T?usp=sharing

	4MC-RT Model 1	**4MC-RT Model 2**	**4MC-RT Model 3**
Samples per class	All	All	All
Masking	False	False	False
Deltas	False	False	False
Truncation	True	True	True
LSTM Width	93	64	32
LSTM Depth	2	2	1
FC Width	0	0	0
FC Depth	0	0	0
Output Layer Width	4	4	4
Learning Rate	0.01	0.001	0.01
N_FFT	2048	512	2048
Mels	40	60	86
Parameters (weights)	190468	113924	37508
Train Accuracy	88.4%	84.0%	82.4%
Validation Accuracy	82.3%	80.3%	80.3%
Test Accuracy	76.8%	76.8%	78.9%

Table 6.15: 3 best models for our 4MC-RT.

	Class 1	**Class 2**	**Class 3**	**Class 4**
Class 1	61	0	0	0
Class 2	2	50	39	0
Class 3	0	0	0	0
Class 4	0	0	0	33

Table 6.16: Numerical Confusion Matrix for 4MC-RT model 1.



(a) 4MC-RT model 1.

(b) 4MC-RT model 2.

Figure 6.9: Graphical Confusion matrixes for four class 4MC-RT model 1 and 4MC-RT model 2.

classifier made us understand that further specialize to 5 classes while using the current dataset was not feasible since the confusion was going to increase.

6.3 Binary Classifier for Excessive Silence (B-S)

In order to detect excessive silence in the middle of phrases, a binary classifier for excessive silence, or B-S was built. Being a different error problem, the previous dataset could not longer be used and, as such, a new one was built. To complete this task, ESPnet's transformer model was used, since it is the one with higher speech quality, to generate a grand total of 4114 samples, which were initially classified as shown in table 6.17. Samples are available on the following Google Drive link[7].

Class description	Number of Samples	Categorical Numerical Class
Clear Audio	2207	1
Excessive Silence	2207	2

Table 6.17: Excessive Silence dataset initial approach.

The neural network was adapted from B-RT since it was also a binary model, where the main difference is the use of a sigmoid activation function (and the associated changes on loss function and accuracy calculation) with only one output node on the last layer. Additionally, the *truncate* variable was always set to *false*, to avoid truncating the error audio before the actual silence, which would cause wrong labels on our system. The 3 best results and the respective architecture can be seen on table 6.18

	B-S Model 1.1	B-S Model 1.2	B-S Model 1.3
Masking	False	False	False
Deltas	False	False	False
Truncation	False	False	False
LSTM Width	64	93	64
LSTM Depth	1	2	2
FC Width	32	0	0
FC Depth	1	0	0
Output Layer Width	1	1	1
Learning Rate	0.01	0.01	0.01
N_FFT	2048	2048	2048
Mels	20	40	76
Parameters (weights)	531777	144895	86657
Train Accuracy	91.4%	91.2%	89.4%
Validation Accuracy	89.0%	87.8%	87.6%
Test Accuracy	92.3%	90.1%	89.1%

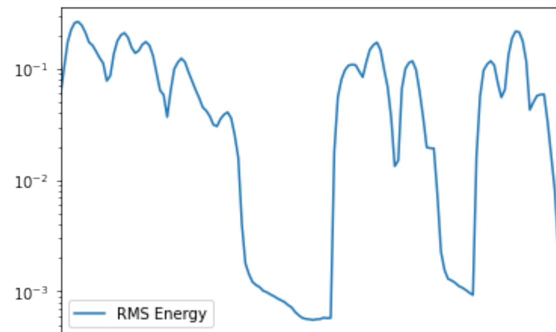
Table 6.18: First iteration of 3 best models for B-S.

While the overall accuracies were not disappointing, improvements could be made, especially on increasing the strength of the given labels, since our initial approach was simply splitting the audios and trusting the transformer synthesizer, as the amount of samples was too large to hear and label manually. To overcome this difficulty, a simple system was developed that would detect the presence of silences automatically on our samples and label them accordingly.

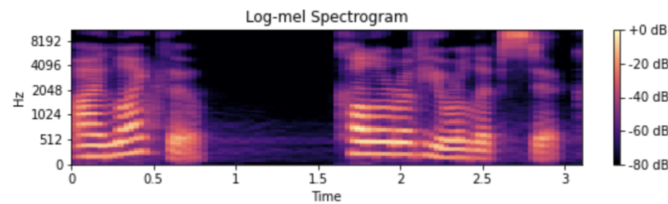
An initial approach was analyzing the overall energy of the signal and search for sudden, prolonged, breaks or increases, as seen on figure 6.10 alongside the corresponding spectrogram, by using the sample energy mean and standard deviation. However, this approach

[7] https://drive.google.com/drive/folders/1rSGrGuRMjHUi5HhYH1oSK-QWDi9aWoQV?usp=sharing

proved to struggle when there was more than one silence segment on any given audio. As an example, if various short silences were present alongside a long one in the same signal, the system would stop considering the long one silence as the energy mean already had a low value and thus it was not detected. In other words, the system was built around the existence of an overall high mean energy. With this in mind, and to avoid the process of reinventing the wheel, research was done and an existing library named *Pydub* was used, which proved to be much simpler and worth regarding implementation complexity and time efficiency. This library used the Decibels relative to full scale (dBFS) scale, where the threshold of -40 dBFS is considered silence. On a final note, an extensive silence was considered artificial if its duration was greater than 0.8 seconds, a value subjected to the consensus of the whole team after hearing various audio samples since the start of the project.



(a) RNS (Root mean square) Energy Variation.



(b) Spectrogram depicting excessive silence with a value bigger than 0.8 seconds.

Figure 6.10: Energy and Spectrogram representation of a signal suffering from excessive silence.

After inputting the previous dataset into the developed system, the output was a stronger dataset, represented in table 6.19, whose results can be analyzed on table 6.20, with the number of the models being 2.x since they were the second iteration of trained models for this classifier.

Class description	Number of Samples	Categorical Numerical Class
Clear Audio	1882	1
Excessive Silence	2532	2

Table 6.19: Improved silence dataset.

The existence of improvements on the performance is evident, with none of the second iteration models going below 91% of accuracy in any set, where the first iteration models went as low as 87.62%. Additionally, and to strengthen this improvement, the best performing model (B-S Model 1.1 for the first iteration and B-S Model 2.1 for the second)

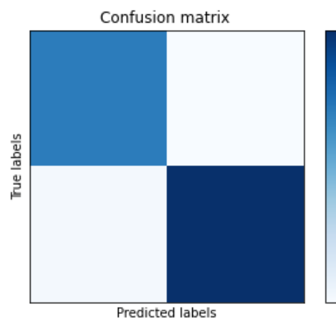
	B-S Model 2.1	B-S Model 2.2	B-S Model 2.3
Masking	False	False	False
Deltas	False	False	False
Truncation	False	False	False
LSTM Width	64	93	64
LSTM Depth	1	2	2
FC Width	32	0	32
FC Depth	1	0	1
Output Layer Width	1	1	1
Learning Rate	0.01	0.01	0.01
N_FFT	2048	2048	512
Mels	20	40	96
Parameters (weights)	531777	144895	635457
Train Accuracy	96.2%	95.3%	94.2%
Validation Accuracy	93.7%	92.6%	91.0%
Test Accuracy	94.3%	91.9%	93.4%

Table 6.20: Second iteration of 3 best models for B-S after improving the dataset labels.

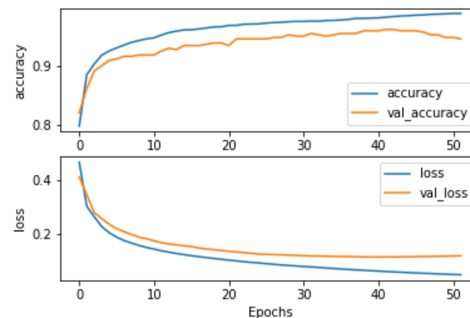
architecture was the same before and after the dataset improvement, having increased the validation set accuracy by 4.71% which implies the original dataset had wrongly classified samples. The numerical and graphical confusion matrix, as well as the loss and accuracy chart across epochs of model B-S model 2.1, can be seen on table 6.21 and figure 6.11.

	Class 1 - Clear Audio	Class 2 - Silence
Class 1	174	10
Class 2	15	243

Table 6.21: Numerical Confusion Matrix for B-S model 2.1.



(a) Confusion Matrix for B-S model 2.1



(b) Accuracy and Loss across epochs for B-S model 2.1

Figure 6.11: Graphical Confusion matrix and evolution of accuracy and loss across epochs for B-S model 2.1.

Looking at the plotted information, it is clear the confusion between excessive silence and good audio is balanced, without neither class being excessively more misclassified than the other. A probable cause for this small, yet existing confusion, could be audios whose silence has a duration of, for example, 0.7 seconds, which are not considered an error by our system since it does not reach our defined threshold. Due to the natural flow of speech,

it is normal for this misinterpretation from both classes to happen, especially if a given sample has more than one phrase, as the existence of punctuation between each may affect this negatively by introducing bigger, yet natural, pauses. Additionally, it can be noted that the loss and accuracy values are evolving progressively and without big differences between the training and test sets, which also increases our confidence on B-S model 2.1. The architecture can be seen in figure 6.12.

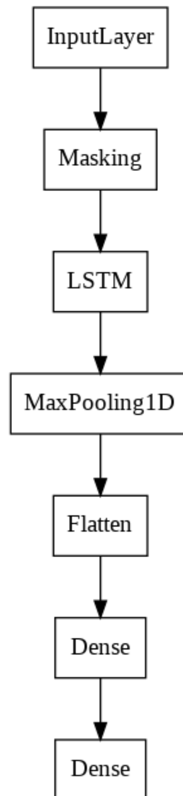


Figure 6.12: Best performing network for B-S

6.4 3 Class Mixed Classifier (3MC-MIX)

The final developed classifier, named 3 class mixed classifier or 3MC-MIX, consists on a mix of the previous classifiers, where three classes were assigned. The first class correspond to the extremely robotic audios, the second class corresponds to the audios with excessive silence and the third class corresponds to clear audio. In other words, the same logic of the 3MC-RT was used, where the second class represented silences instead of slightly robotic samples.

Table 6.22 shows the exact distribution of samples by class. In this case, class two is much more represented than the other two, which makes the dataset unbalanced. However, this fact does not seem to impact the overall final results. Samples can be heard on the following Google Drive link[8].

Table 6.23 shows the best three performing models and their respective values for this classification problem. Overall, the results were considerably good, since we excluded the

[8]https://drive.google.com/drive/folders/1rSGrGuRMjHUi5HhYH1oSK-QWDi9aWoQV?usp=sharing

Class description	Number of Samples	Class
Extremely Robotic	572	1
Excessive Silence	2532	2
Clear	731	3

Table 6.22: Dataset used for representing the 3 classes for 3MC-MIX.

class that introduced confusion in favor of a more easily distinguishable one. In fact, none of the tested architectures had a validation score lower than 90%. As we saw on B-S, the truncation was set to false as we could be excluding the silence segment of our audio while still classifying it with class 2, which would introduce wrong labels on our data.

	3MC-MIX Model 1	3MC-MIX Model 2	3MC-MIX Model 3
Samples per class	All	All	All
Masking	False	False	False
Deltas	False	False	False
Truncation	False	False	False
LSTM Width	64	93	32
LSTM Depth	2	2	1
FC Width	0	0	0
FC Depth	0	0	0
Learning Rate	0.01	0.01	0.01
Output Layer Width	3	3	3
N_FFT	512	2048	2048
Mels	76	40	86
Parameters (weights)	173571	271191	62755
Train Accuracy	96.9%	95.5%	94.1%
Validation Accuracy	93.8%	93.6%	92.9%
Test Accuracy	93.2%	94.0%	93.5%

Table 6.23: 3 best models for 3MC-MIX.

In table 6.24 and image 6.13 the confusion matrix for the best performing architecture, corresponding to 3MC-MIX Model 1, can be seen. It is possible to note that class 3 was the one suffering from a higher degree of misclassification, being especially confused with class 2, which is not a strange result since one of the sources from this third class is the Librispeech dataset whose audios, as we already saw before, try to simulate emotion, being easily misinterpreted as excessive silence. Regarding layer organization in 3MC-MIX model 1, it was identical to the one depicted in figure 6.4.

	Class 1	Class 2	Class 3
Class 1	56	0	1
Class 2	1	244	22
Class 3	2	0	58

Table 6.24: Numerical Confusion Matrix for 3MC-MIX Model 1.

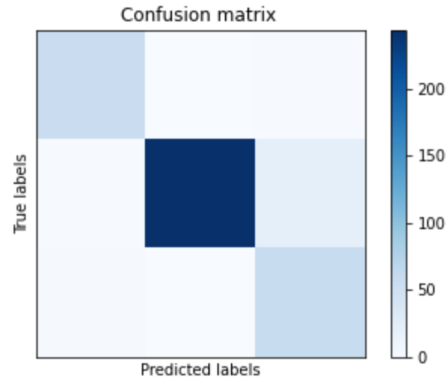


Figure 6.13: Graphical Confusion Matrix for 3MC-MIX Model 1

6.5 Final Overview

In the previous sections we exposed the development process and obtained results while training models for the five classifiers. During the development of the binary classifier for robotic tones B-RT, various architectures were tested culminating in 7 different models that would be used as a base for all future experimentations across all classifiers. Ideally it would be better to test all possible combinations of hyperparameters for each classifier, this was simply not possible due to the internship time constraints. After all the experimentations, the best model for each classifier was chosen as the best overall architecture for the given problem. Table 6.25 shows the overall best scoring model for each classifier side by side.

	B-RT	**3MC-RT2**	**4MC-RT**	**B-S**	**3mc-mix**
Masking	False	False	False	False	False
Deltas	False	False	False	False	False
Truncation	True	True	True	False	False
LSTM Width	64	64	93	64	64
LSTM Depth	2	2	2	1	2
FC Width	0	0	0	32	0
FC Depth	0	0	0	1	0
Output Layer Width	1	3	4	1	3
Learning Rate	0.001	0.001	0.01	0.01	0.01
N_FFT / Window Length	512	512	2048	2048	512
Mels	60	60	40	20	76
Parameters (weights)	77249	101699	190468	531777	173571
Train Accuracy	89.8%	81.7%	88.4%	96.2%	96.9%
Validation Accuracy	93.0%	83.0%	82.3%	93.7%	93.8%
Test Accuracy	96.0%	77.3%	76.8%	94.3%	93.2%

Table 6.25: Comparison of the best model architecture for each classifier.

Looking at the final results, the overall accuracies on all sets for the 5 classifiers can be seen as positive despite having room for improvements. Furthermore, simpler networks seem to perform better across all the classifiers, with B-S having only one LSTM but using an intermediate fully connected layer and B-RT, 3MC-RT, 4MC-RT and 3MC-MIX having LSTMs but lacking the intermediate fully-connected layer, which could mean that using more layers may not be beneficial. Additionally, B-S has a considerably higher number of

parameters but is also the classifier with the biggest dataset, which makes sense.

Regarding the *masking* and *deltas* variables, none of the models made use of them, implying they are not beneficial for our classification task. Avoiding using deltas will actually improve performance regarding training time and data size, since setting this variable to *true* increased the number of features by 3 times. *Truncation* should be used whenever the TTS problem only affects a certain sequence of the audio sample, such as excessive silence.

Focusing on the other variables, there does not seem to exist a direct relationship between their value and the classifier problem. In order to find their ideal value, more experimentations should be made.

This page is intentionally left blank.

Chapter 7

Conclusion

By deciding to tackle the Speech-to-Text (STT) and Text-to-Speech (TTS) fields, Talkdesk is already developing a virtual agent focused on a call-center environment, capable of dealing with simple and repetitive tasks, while real agents deal with complex matters, with the aim of improving and optimizing the overall productivity and reducing downtime. By developing an in-house solution, the company is capable of controlling their solution to a much bigger degree, while lowering the usage costs.

During the initial phase of this internship, we successfully gathered as much knowledge as possible of the field, given the timeframe, which eventually led to changing the original plan of developing a TTS system from the ground up. Alternatively, the option of taking a safer, and more logic, path by improving an already existent one, not only by predicting and avoiding errors in runtime, but also by introducing realistic elements such as office background noise to the generated waveform was chosen. By developing a system with these requirements, it would be possible to improve an existing system.

However, this approach proved to have flaws that would hinder how useful the system would be. With this in mind, the project was once again modified in order to bring more usefulness to the company. Using Google's AutoMOS paper as a source of inspiration, with the final goal changed, now consisting in building various AI models capable of recognizing and classifying errors that could occur on the existing TTS systems such as robotic tone, excessive silence, critical errors, among others. Since the available time for this project was quite limited, more attention was given to the robotic tone and excessive silence problems since they were the easier ones to artificially replicate for building our datasets. In the end, a total of 5 classifiers were developed.

- **Binary Classifier for Robotic Tone** - The binary classifier between clear and robotic tones, used a total of 2024 audios from 6 different sources in order to avoid over-fitting to a single fixed voice while having different tones. Each of these sources has a certain degree of quality. The best performing models for this problem had an accuracy ranging between 88% and 93%, which was slightly below the initial defined quality threshold. However, taking into account the shortness of the dataset as well as the overall bad quality of some samples, the results not having astonishing values were expected. All the work and knowledge gathered by developing this first classifier, was then used when building the next ones.
- **3 Class Classifier for Robotic Tone** - Expanding upon the binary classifier, work went into developing a 3 class classifier, taking into account three different degrees of robotic tones. The dataset was slightly modified and manually labelled in order

to strengthen the associated labels. While the results were positive, with values ranging from 78% to 83%, all trained models proved to be considerably confused between slightly robotic audio and clear audio, which was expected, since the difference between the audios representing these two classes are very slim, with the manually assigned class being somewhat subjective to the human listener.

- **4 Class Classifier for Robotic Tone** - For research purposes, a four class classifier was also made, with the fourth class having only samples from a real audio book. The outcome was interesting, with the trained models achieving good results when classifying extremely robotic audios and the human audiobooks but still getting very confused between the two slightly robotic classes. Overall, the accuracy of the trained models was between 80% and 82%. This proved the developed classifier was being of capable of extracting useful features from the real human voices when compared to the samples generated by an artificial TTS such as Fastspeech or Tacotron. While a 5 class classifier was initially planned, the lack of samples and results made it not worth to test and develop.

- **Binary Classifier for Excessive Silence** - Tackling the excessive silence problem, a binary model was developed, with silences bigger than 0.8 seconds being considered excessive and classified as an error. A new dataset with more than 4000 was built, with the models achieving a validation accuracy value as high as 89%. However, this dataset was optimized using an audio library, resulting in improvements of around 5% on the model's accuracy, with the best model achieving 93.7% accuracy on the validation set. The overall results were good, with a clear distinction being made, and the misclassification being mostly justified by the existence of pauses that naturally occur in speech. This could be improved by making sure each sample only had one sentence, avoiding the existence of punctuation.

- **3 Class Mixed Classifier** - Finally, a mixed model was also developed, consisting on a three class classifier responsible for separating robotic audio, clean audio and audio with excessive silence. While this model inherited most of its characteristics from the 3 Class Classifier for Robotic Tone, the final performance was actually much better, ranging between 93% and 94%, since the class that previously caused the model to be confused was swapped by excessive silence audio, which is much easier to distinguish from the other samples since it has more unique features. Additionally, by being able to distinguish most samples, this model proved that temporal features were being considered, which increased confidence on the chosen Long Short-term memory (LSTM)'s architecture.

While the project's end goal changes did prove to be a challenge at the time, they were actually positive as they allowed the gathering of more information and experience, which would probably be skipped had these changes not been made, allowing a deeper understanding of the older and current speech synthesis processes, systems and techniques. Starting the development of a system with good base knowledge of all the systems involved, gave us a good headstart in identifying certain aspects and possible future difficulties.

The knowledge gathered from the study of TTS fields and from building these models did allow the team to greatly improve, which was evident during the development, especially when going from one model to the other. As an example, the amount of time spent developing the first classifier would be much lower if it had to be done again, since most of the difficulties have already been identified, be it from a practical or theoretical point of view, and the needed solutions found. While the original goal was to develop a system capable of outputting, in a graphical format, the pros and cons of a given TTS system, the

limited time associated with limited resources in terms of hardware and datasets proved to be a big obstacle. The decision to focus on robotic and excessive silence problems proved to be the best option, due to how easy it was to generate samples with these kind of errors when compared to others such as random artifacts. While the initial goal was not completely achieved, the path forged until now allowed us to have proof that our classifiers, with their architectures, were able to achieve good results given our constrained scenario. In other words, the work done can be seen as a proof of concept for the given architectures.

Having already found flaws on our models, such as the confusion with Fastspeech, Tacotron and Transformer on the three and four class models, will allow a much bigger focus on building a strategy to tackle them since it is already known what the problem is.

7.1 Future Work

Taking into consideration the originally planned goal, there are not only improvements to be made on the existing classifiers, possibly correcting their flaws, but also more classifiers to build, tackling problems such critical errors and random artifacts on the speech.

In order to improve upon the existing models, one of the first steps is to considerably increase the size of the existing datasets by more than 10 times. Since the line that distinguishes each class is so thin, the more samples we have, more unique features and patterns can be extracted. Additionally, each dataset should be composed of even more different voice tones. This is important, since if a class is only represented by a single voice, the model will most often than not classify audios based on voice and pitch features instead of specific error features.

The hyperparameters of the current models should also be further tested. As was said previously, only some were tested due to time constraints. More specifically, the pooling layers, the L1 and L2 normalizations, decay value and even batch size were mostly left untouched,assuming the default values depicted on Google's AutoMOS implementation. Changing these values may prove to bring benefits, as the currently they are optimized for Google's specific problem. Additionally, more metric should be recorded, such as precision and recall as they are considerably useful due to the confusion present on the classifiers. While these metrics are easily calculated with the available confusion matrix, an autonomous recording would be an improvement.

In order to improve the training phase, it would also be beneficial to use specific machines instead of Google's Colab platform. The limited Graphics Processing Unit (GPU) usage and sudden breaks when training a model did prove to be problematic and time consuming. The faster and more stable the machine, the more architectural and parametrical combinations can be safely tested, possibly improving the results. In other words, having a more stable machine is something that could improve results in the future.

Finally, once all the error model are developed and as a final goal, a pipeline system should also be made, where a batch of audios from a given TTS system are inputted, inferred by all the different error classifiers and the output is a radar chart, depicting the weaknesses and strengths of said system.

This page is intentionally left blank.

References

Ahire, J. B. (2017). The xor problem in neural networks. Retrieved from https://medium.com/@jayeshbahire/the-xor-problem-in-neural-networks-50006411840b#:~:text=The%20XOr%2C%20or%20%E2%80%9Cexclusive%20or,value%20if%20they%20are%20equal. (Accessed: 2020-06-22)

AI - Natural Language Processing. (n.d.). https://www.tutorialspoint.com/artificial_intelligence/artificial_intelligence_natural_language_processing.htm. Accessed: 2020-01-06.

Alpaydin, E. (2009). *Introduction to machine learning*. MIT press.

Arik, S. Ö., Chrzanowski, M., Coates, A., Diamos, G., Gibiansky, A., Kang, Y., ... Raiman, J. et al. (2017). Deep voice: Real-time neural text-to-speech. In *Proceedings of the 34th international conference on machine learning-volume 70* (pp. 195–204). JMLR.org.

Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Beerends, J. G., Schmidmer, C., Berger, J., Obermann, M., Ullmann, R., Pomy, J., & Keyhl, M. (2013). Perceptual objective listening quality assessment (polqa), the third generation itu-t standard for end-to-end speech quality measurement part i—temporal alignment. *Journal of the Audio Engineering Society*, *61*(6), 366–384.

Bhande, A. (2018). What is underfitting and overfitting in machine learning and how to deal with it. Retrieved from https://medium.com/greyatom/what-is-underfitting-and-overfitting-in-machine-learning-and-how-to-deal-with-it-6803a989c76. (Accessed: 2019-11-20)

Bishop, C. M. (2006). *Pattern recognition and machine learning*. springer.

Capes, T., Coles, P., Conkie, A., Golipour, L., Hadjitarkhani, A., Hu, Q., ... Neeracher, M. et al. (2017). Siri on-device deep learning-guided unit selection text-to-speech system. In *Interspeech* (pp. 4011–4015).

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

Chowdhury, G. G. (2003). Natural language processing. *Annual review of information science and technology*, *37*(1), 51–89.

Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, *12*(Jul), 2121–2159.

Gandhi, R. (2018). Introduction to machine learning algorithms: Logistic regression. Retrieved from https://hackernoon.com/introduction-to-machine-learning-algorithms-logistic-regression-cbdd82d81a36. (Accessed: 2020-01-14)

Gartzman, D. (2019). Getting to know the mel spectrogram. Retrieved from https://towardsdatascience.com/getting-to-know-the-mel-spectrogram-31bca3e2d9d0. (Accessed: 2019-11-18)

Gibiansky, A., Arik, S., Diamos, G., Miller, J., Peng, K., Ping, W., . . . Zhou, Y. (2017). Deep voice 2: Multi-speaker neural text-to-speech. In *Advances in neural information processing systems* (pp. 2962–2970).

Grancharov, V., Zhao, D. Y., Lindblom, J., & Kleijn, W. B. (2006). Non-intrusive speech quality assessment with low computational complexity. In *Ninth international conference on spoken language processing*.

Griffin, D., & Lim, J. (1984). Signal estimation from modified short-time fourier transform. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, *32*(2), 236–243.

Gupta, P. (2017). Regularization in machine learning. Retrieved from https://towardsdatascience.com/regularization-in-machine-learning-76441ddcf99a#:~:text=increasing%20model%20interpretability.-,Regularization,linear%20regression%20looks%20like%20this.. (Accessed: 2020-04-20)

Harrington, P. (2012). *Machine learning in action*. Manning Publications Co.

Harrison, O. (2018). Machine learning basics with the k-nearest neighbors algorithm. Retrieved from https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761. (Accessed: 2020-01-08)

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), 1735–1780.

Hsu, W.-N., Zhang, Y., Weiss, R. J., Zen, H., Wu, Y., Wang, Y., . . . Shen, J. et al. (2018). Hierarchical generative modeling for controllable speech synthesis. *arXiv preprint arXiv:1810.07217*.

Ito, K. (2017). The lj speech dataset. https://keithito.com/LJ-Speech-Dataset/.

Jordan, J. (2018). Setting the learning rate of your neural network. Retrieved from https://www.jeremyjordan.me/nn-learning-rate/. (Accessed: 2020-06-24)

Jurafsky, D., & Martin, J. H. (2019). *Speech and langauge processing*.

Kalchbrenner, N., Elsen, E., Simonyan, K., Noury, S., Casagrande, N., Lockhart, E., . . . Kavukcuoglu, K. (2018). Efficient neural audio synthesis. *arXiv preprint arXiv:1802.08435*.

Kim, D.-S. (2005). Anique: An auditory model for single-ended speech quality estimation. *IEEE Transactions on Speech and Audio Processing*, *13*(5), 821–831.

Kingma, D. P. [Diederik P], & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kingma, D. P. [Durk P], & Dhariwal, P. (2018). Glow: Generative flow with invertible 1x1 convolutions. In *Advances in neural information processing systems* (pp. 10215–10224).

Kingma, D. P. [Durk P], Salimans, T., Jozefowicz, R., Chen, X., Sutskever, I., & Welling, M. (2016). Improved variational inference with inverse autoregressive flow. In *Advances in neural information processing systems* (pp. 4743–4751).

Kittur, A., Chi, E. H., & Suh, B. (2008). Crowdsourcing user studies with mechanical turk. In *Proceedings of the sigchi conference on human factors in computing systems* (pp. 453–456). ACM.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (pp. 1097–1105).

Kubichek, R. (1993). Mel-cepstral distance measure for objective speech quality assessment. In *Proceedings of ieee pacific rim conference on communications computers and signal processing* (Vol. 1, pp. 125–128). IEEE.

LeCun, Y., Bottou, L., Bengio, Y., Haffner, P. et al. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324.

Librosa - STFT. (n.d.). https://librosa.org/librosa/generated/librosa.core.stft.html. Accessed: 2020-06-22.

Lo, C.-C., Fu, S.-W., Huang, W.-C., Wang, X., Yamagishi, J., Tsao, Y., & Wang, H.-M. (2019). Mosnet: Deep learning based objective assessment for voice conversion. *arXiv preprint arXiv:1904.08352*.

Malfait, L., Berger, J., & Kastner, M. (2006). P. 563—the itu-t standard for single-ended speech quality assessment. *IEEE Transactions on Audio, Speech, and Language Processing*, *14*(6), 1924–1934.

Marcus, M., Santorini, B., & Marcinkiewicz, M. A. (1993). Building a large annotated corpus of english: The penn treebank.

Mel Scale. (n.d.). https://www.wikiwand.com/en/Mel_scale. Accessed: 2019-11-18.

Mishra, A. (2018). Metrics to evaluate your machine learning algorithm. Retrieved from https://medium.com/thalus-ai/performance-metrics-for-classification-problems-in-machine-learning-part-i-b085d432082b. (Accessed: 2020-01-18)

Morise, M., Yokomori, F., & Ozawa, K. (2016). World: A vocoder-based high-quality speech synthesis system for real-time applications. *IEICE TRANSACTIONS on Information and Systems*, *99*(7), 1877–1884.

Nielsen, M. A. (2015). *Neural networks and deep learning*. Determination press San Francisco, CA, USA:

Olah, C. (2015). Understanding lstm networks. Retrieved from https://colah.github.io/posts/2015-08-Understanding-LSTMs/. (Accessed: 2019-11-18)

Oord, A. v. d., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., . . . Kavukcuoglu, K. (2016). Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*.

Oord, A. v. d., Li, Y., Babuschkin, I., Simonyan, K., Vinyals, O., Kavukcuoglu, K., ... Stimberg, F. et al. (2017). Parallel wavenet: Fast high-fidelity speech synthesis. *arXiv preprint arXiv:1711.10433*.

Panayotov, V., Chen, G., Povey, D., & Khudanpur, S. (2015). Librispeech: An asr corpus based on public domain audio books. In *2015 ieee international conference on acoustics, speech and signal processing (icassp)* (pp. 5206–5210). IEEE.

Patton, B., Agiomyrgiannakis, Y., Terry, M., Wilson, K., Saurous, R. A., & Sculley, D. (2016). Automos: Learning a non-intrusive assessor of naturalness-of-speech. *arXiv preprint arXiv:1611.09207*.

Ping, W., Peng, K., & Chen, J. (2018). Clarinet: Parallel wave generation in end-to-end text-to-speech. *arXiv preprint arXiv:1807.07281*.

Ping, W., Peng, K., Gibiansky, A., Arik, S. O., Kannan, A., Narang, S., ... Miller, J. (2017). Deep voice 3: Scaling text-to-speech with convolutional sequence learning. *arXiv preprint arXiv:1710.07654*.

Prenger, R., Valle, R., & Catanzaro, B. (2019). Waveglow: A flow-based generative network for speech synthesis. In *Icassp 2019-2019 ieee international conference on acoustics, speech and signal processing (icassp)* (pp. 3617–3621). IEEE.

Pupale, R. (2018). Support vector machines(svm) — an overview. Retrieved from https://towardsdatascience.com/https-medium-com-pupalerushikesh-svm-f4b42800e989. (Accessed: 2020-01-08)

Ren, Y., Ruan, Y., Tan, X., Qin, T., Zhao, S., Zhao, Z., & Liu, T.-Y. (2019). Fastspeech: Fast, robust and controllable text to speech. In *Advances in neural information processing systems* (pp. 3165–3174).

Ribeiro, F., Florêncio, D., Zhang, C., & Seltzer, M. (2011). Crowdmos: An approach for crowdsourcing mean opinion score studies. In *2011 ieee international conference on acoustics, speech and signal processing (icassp)* (pp. 2416–2419). IEEE.

Richards, J. C., & Schmidt, R. W. (2013). *Longman dictionary of language teaching and applied linguistics*. Routledge.

Rising, L., & Janoff, N. S. (2000). The scrum software development process for small teams. *IEEE software*, *17*(4), 26–32.

Rix, A. W., Beerends, J. G., Hollier, M. P., & Hekstra, A. P. (2001). 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, pp. 749–752). IEEE.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *nature*, *323*(6088), 533–536.

Saha, S. (2018). A comprehensive guide to convolutional neural network. Retrieved from https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53. (Accessed: 2019-11-18)

Sampling (signal processing). (n.d.). https://en.wikipedia.org/wiki/Sampling_(signal_processing). Accessed: 2020-06-26.

Sarkar, D. (2016). *Text analytics with python*. Springer.

Shah, T. (2017). About train, validation and test sets in machine learning. Retrieved from https://towardsdatascience.com/train-validation-and-test-sets-72cb40cba9e7. (Accessed: 2019-11-20)

Shen, J., Pang, R., Weiss, R. J., Schuster, M., Jaitly, N., Yang, Z., ... Skerrv-Ryan, R. et al. (2018). Natural tts synthesis by conditioning wavenet on mel spectrogram predictions. In *2018 ieee international conference on acoustics, speech and signal processing (icassp)* (pp. 4779–4783). IEEE.

Srivastava, R. K., Greff, K., & Schmidhuber, J. (2015). Highway networks. *arXiv preprint arXiv:1505.00387*.

Sunasra, M. (2017). Performance metrics for classification problems in machine learning. Retrieved from https://medium.com/thalus-ai/performance-metrics-for-classification-problems-in-machine-learning-part-i-b085d432082b. (Accessed: 2020-01-18)

Sutskever, I., Vinyals, O., & Le, Q. (2014). Sequence to sequence learning with neural networks. *Advances in NIPS*.

Thanaki, J. (2017). *Python natural language processing*. Packt Publishing Ltd.

Turing, A. M. (2009). Computing machinery and intelligence. In *Parsing the turing test* (pp. 23–65). Springer.

Umesh, S., Cohen, L., & Nelson, D. (1999). Fitting the mel scale. In *1999 ieee international conference on acoustics, speech, and signal processing. proceedings. icassp99 (cat. no. 99ch36258)* (Vol. 1, pp. 217–220). IEEE.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems* (pp. 5998–6008).

Vieira, M. (2018/2019). Project management, risk management. Universidade de Coimbra.

Wagstaff, K., Cardie, C., Rogers, S., Schrödl, S. et al. (2001). Constrained k-means clustering with background knowledge. In *Icml* (Vol. 1, pp. 577–584).

Wang, X. (2019). Tutorial on end-to-end text-to-speech synthesis: Part 1 – neural waveform modeling. Retrieved from https://www.slideshare.net/jyamagis/tutorial-on-endtoend-texttospeech-synthesis-part-1-neural-waveform-modeling. (Accessed: 2019-10-10)

Wang, Y., Skerry-Ryan, R., Stanton, D., Wu, Y., Weiss, R. J., Jaitly, N., ... Bengio, S. et al. (2017). Tacotron: Towards end-to-end speech synthesis. *arXiv preprint arXiv:1703.10135*.

Wang, Y., Stanton, D., Zhang, Y., Skerry-Ryan, R., Battenberg, E., Shor, J., ... Saurous, R. A. (2018). Style tokens: Unsupervised style modeling, control and transfer in end-to-end speech synthesis. *arXiv preprint arXiv:1803.09017*.

Wiegers, K., & Beatty, J. (2013). *Software requirements*. Pearson Education.

Xu, B., Wang, N., Chen, T., & Li, M. (2015). Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*.

Xu, D., & Tian, Y. (2015). A comprehensive survey of clustering algorithms. *Annals of Data Science*, *2*(2), 165–193.

Yamagishi, J., Nose, T., Zen, H., Ling, Z.-H., Toda, T., Tokuda, K., … Renals, S. (2009). Robust speaker-adaptive hmm-based text-to-speech synthesis. *IEEE Transactions on Audio, Speech, and Language Processing*, *17*(6), 1208–1230.

Yiu, T. (2019). Understanding random forest. Retrieved from https://towardsdatascience.com/understanding-random-forest-58381e0602d2. (Accessed: 2020-01-14)

YourDictionary. (2019). Machine-learning. (n.d.) https://www.yourdictionary.com/machine-learning/.