DEVELOPMENT OF AN END-TO-END AUTOMATIC SPEECH RECOGNITION SYSTEM USING CONNECTIONIST TEMPORAL CLASSIFICATION FOR THE TSHIVENDA LANGUAGE

by

JONAS MOSWEU MEHLAPE

DISSERTATION

Submitted in fulfilment of the requirements for the degree of

MASTERS OF SCIENCE

in

COMPUTER SCIENCE

in the

FACULTY OF SCIENCE AND AGRICULTURE

(School Of Mathematical and Computer Sciences)

at the

UNIVERSITY OF LIMPOPO

Supervisor: Prof TI Modipa

2025

DEDICATION

This dissertation is dedicated to my family for their steadfast support and encouragement throughout this journey. Your love and faith in me have been my most significant source of inspiration and strength.

I would also like to dedicate this research to my supervisor, Dr. Modipa, for your invaluable guidance, patience, and wisdom, which has played a pivotal role in shaping this project. Your mentorship has inspired me to push the boundaries of my capabilities.

Finally, the Tshivenda-speaking community and all under-resourced language communities may contribute to preserving your rich linguistic heritage for future generations.

DECLARATION OF AUTHORSHIP

I, Jonas Mosweu Mehlape, hereby declare that development of an end-to-end automatic speech recognition system using connectionist temporal classification for Tshivenda language is my original work. Any information derived from other sources has been properly acknowledged through full reference. Additionally, I declare that this work is original and has not been presented to any other university for the purpose of obtaining a degree or certification.

Signature:  Date: 28-03-2025

ACKNOWLEDGEMENTS

I extend my heartfelt gratitude to my supervisor, Dr. Modipa, for your unwavering support, insightful guidance, and remarkable patience throughout this research journey. Your expertise and mentorship have played a crucial role in the successful completion of this work, and I am deeply appreciative of the growth opportunities you have provided.

I am equally thankful to my family for their endless love, encouragement, and confidence in my abilities. Your unwavering support has been the cornerstone of my strength during this demanding yet fulfilling process.

Finally, I sincerely appreciate my friends and colleagues for their uplifting words, constructive feedback, and companionship, which have made this journey more enjoyable and manageable.

To everyone who has contributed to this endeavor, thank you wholeheartedly.

ABSTRACT

This study centers on creating an automatic speech recognition (ASR) system for Tshivenda, one of South Africa's under-resourced languages. Utilizing the Connectionist Temporal Classification (CTC) framework and the NCHLT speech corpus.

This study focuses on developing an E2E ASR system leveraging CTC techniques for the Tshivenda language. The primary objective is to develop and evaluate an ASR system for the Tshivenda language using the CTC approach. This involves designing and training an ASR model using the NCHLT speech corpus, optimizing model performance through hyperparameter tuning (e.g., learning rate, dropout rate), and evaluating the system's accuracy through essential metrics such as WER and training loss. The research also focuses on identifying key challenges in recognizing Tshivenda speech and proposes improvements for future work in this area.

However, there are several delimitations to the scope of the study that should be considered. First, the research relies on the NCHLT speech corpus, which, although valuable, has limited dialectal diversity and does not fully represent all regional variations of Tshivenda. Additionally, the model was primarily trained on clean speech data, and as such, it does not extensively address the challenges of handling noisy environments or spontaneous speech. Furthermore, while the study focuses on a CTC-based deep learning model, it does not explore the integration of external language models, such as transformer-based models, which could further enhance performance. Finally, due to hardware limitations, the model was trained for 30 epochs, which may have constrained the model's ability to reach its optimal performance, potentially impacting the accuracy of the final system.

The model's performance was assessed over 30 epochs using essential metrics, including Word Error Rate (WER), training loss, and validation loss. The top-performing model achieved a final WER of 0.3934, highlighting notable advancements in Tshivenda speech recognition.

This research highlights the promise of deep learning models in creating ASR systems for under-resourced languages, while also pointing out critical directions for future

exploration. Key advancements include expanding the dataset, integrating language models, and improving the model's resilience to noisy conditions and spontaneous speech. These steps are essential for enhancing accuracy and practical usability. The study contributes to the broader mission of promoting language preservation and accessibility through technological innovation.

***Keywords*:** Automatic Speech Recognition, Tshivenda, Under-Resourced Languages, Connectionist Temporal Classification, Convolutional Neural Networks, Recurrent Neural Networks

Table of Contents

Contents

DEDICATION i

DECLARATION OF AUTHORSHIP ii

ACKNOWLEDGEMENTS iii

ABSTRACT iv

Table of Contents vi

List of Figures ix

List of Tables x

Chapter 1: Introduction 1

1.1 Problem Statement 1

1.2 Motivation 2

1.3 Methodology and Analytical Procedures 5

1.4 Dissertation Outline 7

Chapter 2: Literature Review 9

2.1 Introduction 9

2.2 South African Languages 10

2.2.1 Pronunciation modelling 11

2.2.2 Acoustic Models 13

2.2.3 Language Models 20

2.3 End-to-End Architecture for Speech Recognition 21

2.3.1 Connectionist Temporal Classification 22

2.3.2 Attention 26

2.4 Tshivenda Language 26

2.5 Conclusion 28

Chapter 3: Methodology 29
3.1 Introduction 29
3.2 Tshivenda Speech Corpus 29
3.3 Model Development 34
3.3.1 Tshivenda Speech Dataset 34
3.3.2 Text Data Preprocessing 35
3.3.3 Tshivenda Dataset Objects 35
3.3.4 Visualization of the Speech Data 35
3.3.5 Connectionist Temporal Classification – based Model Training 37
3.3.6 Inference 40
3.4 Evaluation 40
3.4.1 Connectionist Temporal Classification Loss 40
3.4.2 Word Error Rate 41
3.5 Conclusion 42
Chapter 4: Experimental Results 43
4.1 Introduction 43
4.2 Optimised Hyperparameter Model 43
4.3 All Experiments 51
4.4 Conclusion 54
Chapter 5: Conclusion 55
5.1 Introduction 55
5.2 Summary of Results 55
5.3 Future Work 56
5.3.1 Expanding the Dataset 56
5.3.2 Incorporating Language Models 57
5.3.3 Handling Noisy and Spontaneous Speech 57

5.3.4 Exploring Multimodal ASR Systems 57
5.4 Limitations 57
5.4.1 System Constraints: Limitation on Training Epochs 58
5.4.2 Limited Dataset Size and Diversity 58
5.4.3 Sensitivity to Noise and Speaker Variability 58
Bibliography 59

List of Figures

Figure 1.1: End-to-End Model

Figure 2.1: Historical Development of Automatic Speech Recognition

Figure 2.2: Recurrent Neural Network Architecture

Figure 2.3: End-to-End Architecture

Figure 2.4: Connectionist Temporal Classification Architecture

Figure 3.1: Training Flow Chart

Figure 3.2: Words Frequency Graph for Training Set

Figure 3.3: Words Frequency Graph Test Set

Figure 3.4: Words Frequency Graph Validation Set

Figure 3.5: Spectrogram

Figure 3.6: Signal Wave

Figure 4.1: Training and Validation Loss per Epoch

Figure 4.2: Word Error Rate per Epoch

Figure 4.3: Optimised Hyperparameter Model's Target and Prediction Results

List of Tables

Table 3.1: Summary of the clean Tshivenda corpora

Table 3.2: Training and testing set

Table 4.1: Optimised Hyperparameter

Table 4.2: Optimised Hyperparameter Model's Target and Prediction Results

Chapter 1: Introduction

1.1 Problem Statement

Automatic Speech Recognition (ASR) plays a vital role in enhancing both human-to-human and human-computer interactions (Yu, 2016). Speech is the primary mode of communication among humans, and the development of ASR stems from both technological curiosity about replicating human speech abilities in machines and the need to automate tasks involving human-machine interaction (Juang, 2005). ASR technology holds significant promise across various applications, such as voice input and speech search, which can simplify everyday tasks and improve convenience (Hayashi, 2018). However, challenges in ASR persist, including low accuracy and incorrect transcriptions.

Automatic Speech Recognition (ASR) technology emerged in the mid-1970s with the introduction of hidden Markov models (HMM), which utilized probability functions to accurately transcribe words. HMM became a widely used approach in machine learning for modeling sequences, such as speech (Miller, 1999). By 2011, two prominent ASR techniques had gained popularity: Tandem and the Deep Neural Network (DNN)-HMM hybrid. The DNN-HMM method, which replaced the Gaussian Mixture Model (GMM) with a DNN in acoustic modeling, became the standard practice. In 2016, end-to-end (E2E) speech recognition emerged as a groundbreaking approach, rapidly advancing ASR performance. The E2E methodology directly maps input acoustic features to output sequences, optimizing system evaluation criteria such as Word Error Rate (WER).

The end-to-end (E2E) approach includes the Connectionist Temporal Classification (CTC) method, a training framework for tasks like speech recognition and handwriting identification. CTC is particularly effective when the alignment between input and output is unknown, as it aligns audio features with their corresponding transcript characters. This scalable solution has been applied successfully to monotonic sequence transduction tasks, such as handwriting recognition (Salazar, 2019). CTC efficiently maps audio features to their textual counterparts.

This study focuses on developing an E2E ASR system leveraging CTC techniques for the Tshivenda language. The objective is to assess its performance in addressing the needs of Tshivenda, an under-resourced Indigenous African language spoken in South Africa.

1.2 Motivation

The integration of deep neural networks (DNNs) and recurrent neural networks (RNNs) has greatly advanced automatic speech recognition (ASR), particularly in acoustic and language modeling (Audhkhasi, 2017). However, these models are computationally intensive and require extensive datasets. To remain effective, they must also be adaptable to real-world scenarios that evolve over time (Chang, 2021).

Building on the progress made in automatic speech recognition (ASR) with DNNs and RNNs, additional techniques like Connectionist Temporal Classification (CTC) and Convolutional Neural Networks (CNNs) provide specialized solutions for enhancing sequence modeling and feature extraction. While DNNs and RNNs are essential for acoustic and language modeling, CTC offers a framework for alignment-free training, which is crucial for efficiently processing unsegmented audio data. This discussion will examine how each approach—from the sequence management in RNNs to the feature extraction power of CNNs—contributes to meeting the complex requirements of ASR systems, ultimately enabling more accurate and adaptable applications in real-world scenarios.

David Rumelhart introduced the concept of the Recurrent Neural Network (RNN) in 1986, building on John Hopfield's rediscovery of Hopfield networks in 1982, which are a specific type of RNN. RNNs are widely used to process sequential data, and over the years, numerous modifications have been proposed to overcome their limitations, enabling RNNs to achieve impressive performance (Soullard, 2019). The input and output representations of an RNN extend beyond basic models, with the internal state allowing for the development of time series models that can be trained discriminatively (Graves, 2006). Additionally, RNNs can be expanded to include Bidirectional Recurrent Neural Networks (BRNNs), which allow for training without being limited to using only past input data, providing access to future input as well (Schuster, 1997).

CNNs are widely recognized in machine learning, particularly in computer vision, due to their exceptional performance in image classification (Koundinya, 2018). CNNs consist of multiple 2D convolutions, which contribute to over 90% of the total computation (Chang, 2016). In the context of speech processing, a 2D CNN network has been developed to capture both local and global emotion-related features from voice data and Mel spectrograms (Zhao, 2019). This type of model focuses on extracting spectral information by considering the spatial relationships within the image's channels. A 2D CNN-based approach offers comparable performance to other methods while requiring significantly less computational power (Koundinya, 2018).

CTC was introduced in 2006 to describe the output and scoring process of a neural network. It provides a significant advantage by eliminating the need for pre-segmentation of training data and post-processing to convert the output of a RNN into a sequence of labels. The CTC method incorporates an additional unit to represent the probability of a "blank" label, indicating the absence of a class label (Soullard, 2019). This approach enables neural networks to be trained with pairs of input data and target labels (text). The network is designed to output labels using a specific encoding scheme (Scheidl, 2018). The CTC loss function allows for training neural networks on sequence-to-sequence tasks without requiring prior alignment of input and output sequences. Traditionally, CTC has been employed for sequential, single-label problems, where each element in the sequence is assigned only one class (Wigington, 2019).

A fast method is required to compute the conditional probability of individual labels, which can be complex due to the large number of possible pathways associated with each labeling. Salazar (2019) highlighted this as a significant challenge. To address this, the forward-backward algorithm, commonly used in Hidden Markov Models (HMMs), can be applied through dynamic programming (Rabiner, 1989). The key concept is to decompose the annotation by recursively adding paths that align with the prefixes of the labeling. By using forward and backward variables in a recursive manner, these iterations can be computed efficiently.

It was analyzed that a label sequence modified with a blank space appended on both the start and end of the labels, and placing blank spaces between every occurrence of a pair

of labels, will ensure that the blanks follow output routes. As a result, the length of the label sequence becomes twice the label length plus one. The calculation of the probabilities of the appended prefixes of the label sequences provides movement between both the blank and non-blank labels. The prefixes may begin with a blank space.

The CTC and mask CTC are normally used for the non-autoregressive E2E ASR (Higuchi, 2020). These approaches were used in three settings, each with a different language and amount of data for training. The first setting was the 81-hour Wall Street Journal in the English language. The second is the 581-hour data consisting of continuous speech in the Japanese language. Finally, the 16-hour Voforge speech data using the Italian language. The results showed that the outcome of the mask CTC approach performed better than the standard CTC model for non-autoregressive models. The conclusion was that the mask CTC model can outperform the ordinary CTC model while maintaining decoding speed.

CTC has also been applied to recognize code-switched speech. When combined with a language identification (LID) system trained on monolingual data, the CTC approach produced significant results. The models, "LID3" and "LID4," were trained using DNN and LSTM on monolingual data, while "LID5" and "LID6" were trained with code-switched data. Among them, the LID6 model, which was trained with code-switched data, achieved the highest accuracy on the code-switched test set, with a Word Error Rate (WER) of 28.06%. For the Chinese language, the WER was 13.23%. The study concluded that evaluating the data prior to development could result in a relative improvement of up to 4.2%.

CTC and attention-based architectures are both utilized for end-to-end (E2E) speech recognition (Watanabe, 2017). In the experiments, the trials employed the WSJ1 and WSJ0 clean speech corpora, along with the CHiME-4 noisy speech corpus. The study demonstrated that the hybrid CTC/attention architecture outperformed the attention-based E2E ASR by addressing misalignment issues. Unlike traditional Japanese and Mandarin Chinese ASR systems, which require linguistic resources like a morphological analyzer, pronunciation dictionary, and language model, this hybrid approach eliminates the need for such resources.

CTC is a training technique commonly used with encoder-based models in speech recognition tasks, especially when there is no alignment between input and output sequences. In the context of automatic speech recognition, the model produces a probability distribution over all possible output symbols at each timestep, and CTC decodes this by using a process that can handle the misalignment between speech and text. The CTC loss function is trained to minimize the difference between the predicted sequence and the true sequence, and the decoding process involves selecting the most probable output sequence based on the cumulative probability of each output symbol across time.

1.2.1 Aim

The study aims to develop an end-to-end automatic speech recognition system for the Tshivenda language using the connectionist temporal classification approach.

1.2.2 Objectives

The objectives of the study are to:

I. Perform exploratory data analysis on the Tshivenda speech corpus.
II. Develop the end-to-end speech recognition system using the CTC technique.
III. Evaluate the speech recognition system accuracy.

1.2.3 Research Questions

I. What are the best approaches to analyse and explore the speech corpus?
II. How to optimally develop an end-to-end automatic speech recognition system?
III. What is the accuracy of the Tshivenda end-to-end automatic speech recognition system?

1.3 Methodology and Analytical Procedures

1.3.1 Data Set

This study used the National Centre for Human Language Technology (NCHLT) speech corpus, which includes wide-band speech from 200 individuals across each of South Africa's eleven official languages. Tshivenda was chosen as the focus language among the twelve official languages. The Tshivenda speech corpus consists of approximately 50

hours of orthographically transcribed speech in Tshivenda (Bernard, 2014), along with pronunciation dictionaries containing a total of 15,000 words.

1.3.2 Connectionist Temporal Classification Framework

The CTC-based ASR system will be built using the Python library TensorFlow, which is open-source. The development process will start with defining a 2D CNN comprising two layers, with the output subsequently passed into an RNN. The optimization algorithm used will be ReLU (Rectified Linear Unit). To enhance model performance, various hyperparameters will be tuned, including the number of epochs and the dropout rate. Additionally, different activation functions will be tested to determine the best configuration for the ASR system. While the SoftMax activation function is typically applied to well-resourced languages, its suitability for under-resourced languages will also be explored.

1.3.3 Evaluation of the End-to-End model

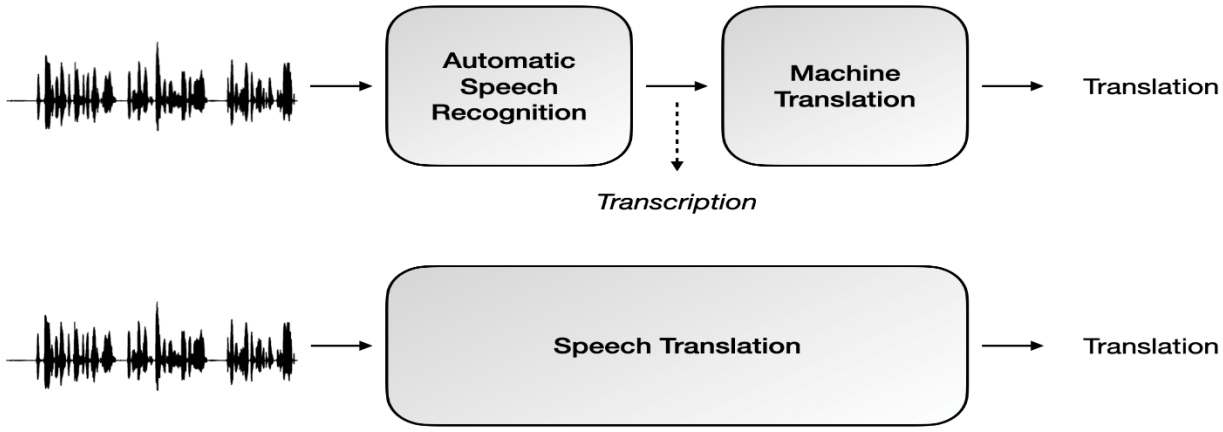

Figure 1 .1 End-to-End Model

Figure 1.1 depicts two different approaches to speech translation. The first approach, known as the two-stage process, involves Automatic Speech Recognition (ASR) followed by Machine Translation (MT). In this method, the input is an audio waveform representing speech, which is first processed by the ASR system to produce a textual transcription in the source language. ASR consists of several steps, including signal processing to reduce noise, acoustic modeling to convert the audio into phonetic units, language modeling to predict the most likely sequence of words, and decoding to generate the final transcription.

Once the transcription is obtained, it is passed to the MT system, which translates the text into the target language by analyzing grammatical structures and semantic meaning.

The second approach, end-to-end speech translation, directly translates the speech waveform into the target language text without generating an intermediate transcription. This method integrates both ASR and MT into a single system, using neural machine translation models to process the audio input and generate the corresponding translation. While this approach has advantages in terms of efficiency and reduced error propagation, we have chosen the two-stage approach due to its modularity and ease of debugging. Separating ASR and MT allows for better control over each stage, making it easier to improve and fine-tune the individual components, ensuring higher accuracy and reliability in speech translation.

The Word Error Rate (WER) is the key metric used to evaluate the performance of speech recognition systems. It assesses the frequency of errors in the predicted word sequences by examining the occurrences of insertions, substitutions, and deletions. WER has been effectively used to measure these types of errors (Barnard, 2014). However, local error estimates do not fully capture the broader error patterns, such as the total number of substitutions within a sentence. To calculate WER, the total number of substitutions, insertions, and deletions in a set of recognized words is added up, and this total is subtracted from the total number of words spoken.

1.4 Dissertation Outline

This dissertation is outlined as follows:

- Chapter 2 reviews relevant literature, examining prior research and theoretical frameworks that underpin the study. It also identifies gaps in existing research and situates the current study within the broader field.
- Chapter 3 outlines the research methodology, detailing the data collection and analysis techniques used, as well as the design and development of the ASR system. This includes the application of the CTC method and the utilization of the NCHLT speech corpus.

- Chapter 4 presents and analyzes the results of the study. It evaluates the performance of the ASR system in the context of the research objectives and hypotheses.
- Chapter 5 concludes the dissertation by summarizing the main findings, discussing their implications for the field, and suggesting avenues for future research.

Chapter 2: Literature Review

2.1 Introduction

In this chapter, we will focus on the literature review by examining books and scholarly papers related to the development of an end-to-end ASR system using CTC for the Tshivenda language, as well as the relevant research and theories in this field. This chapter will address key aspects of ASR research, including pronunciation modelling, acoustic models, language models, and an overview of E2E systems. Additionally, it will provide a discussion on the Tshivenda language itself and its role in ASR development. n this chapter,

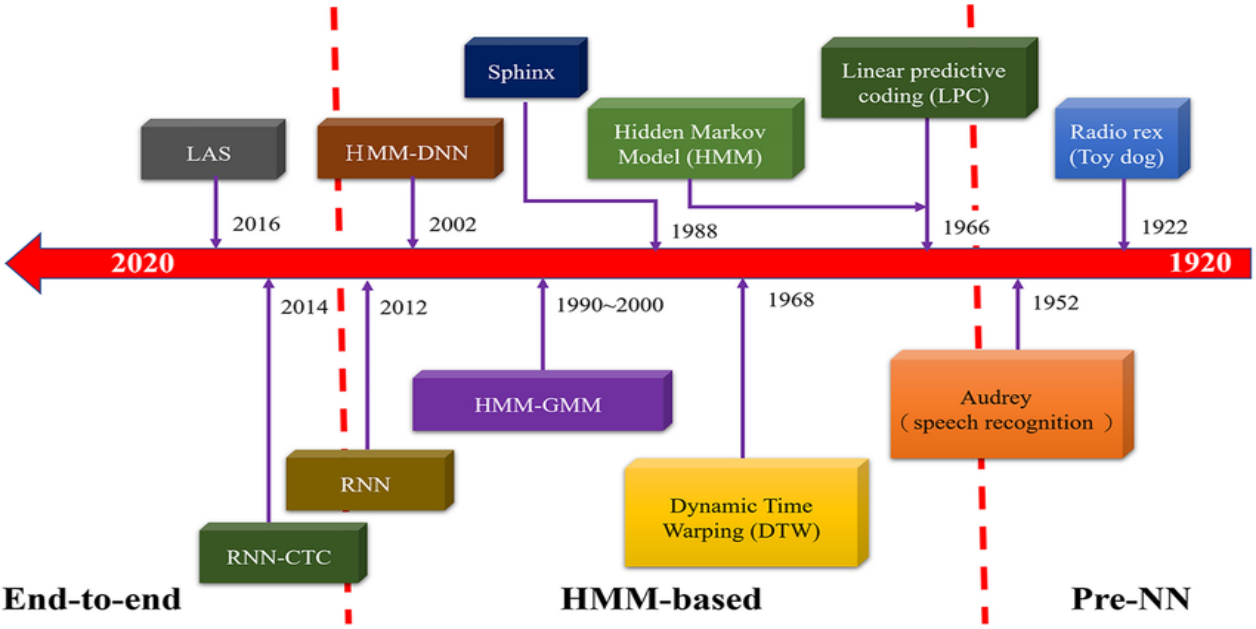

Figure 2.1 Historical Development of Automatic Speech Recognition (Sung, 2023)

Figure 2.1 presents a timeline illustrating the historical development of Automatic Speech Recognition (ASR), highlighting key milestones and technological advancements from 1920 to 2020 (Sung, 2023). The timeline categorizes these developments into three major phases: "Pre-NN" (pre-neural network), "HMM-based" (Hidden Markov Model-based), and modern "End-to-End" methods. Early experiments, such as the "Radio Rex" toy (1922) and Audrey (1952), marked the initial steps in speech recognition. Linear Predictive Coding (LPC) emerged in 1966, followed by Dynamic Time Warping (DTW) in 1968, which became a foundational alignment technique. The 1980s and 1990s saw the dominance of Hidden Markov Models (HMMs), with the HMM-GMM (Gaussian Mixture Model)

framework becoming a standard approach between 1990 and 2000. Notable HMM-based systems like Sphinx (1988) contributed to the field, and the introduction of HMM-DNN (Deep Neural Network) architectures in 2002 improved ASR accuracy by integrating neural networks into traditional methods.

The shift toward end-to-end speech recognition began around 2012 with the emergence of Recurrent Neural Networks (RNNs), leading to the introduction of RNN-CTC (Connectionist Temporal Classification) in 2014. This innovation enabled direct mapping from acoustic input to textual output, simplifying the ASR pipeline by eliminating intermediate modeling steps. Further advancements, such as Listen, Attend, and Spell (LAS) in 2016, refined the end-to-end approach. The timeline underscores RNN-CTC as a pivotal development, marking a transition from HMM-based systems that relied on complex manual feature engineering to data-driven neural architectures. RNN-CTC and similar models provided advantages such as simplified system design, improved data efficiency, and enhanced accuracy, particularly in noisy environments. The timeline's dashed lines indicate that these advancements were not strictly linear, as multiple approaches were explored simultaneously.

2.2 South African Languages

ASR is a technology that enables users to interact with information systems using spoken language instead of traditional input methods like keypads (Malik, 2021). This technology is widely utilized in applications such as call routing and information retrieval. Given that speech is the most natural, efficient, and favored form of human communication, it is logical to anticipate that many would prefer voice input over conventional tools like keyboards. ASR supports this preference by translating spoken language into text, often in the corresponding language's script, using audio recordings or real-time speech captured via a microphone (Bouadjenek, 2021).

An optimal ASR system would precisely decode spoken input, recognize the words, and use them as instructions for a computer to execute specific tasks. ASR has been a key area of research in machine learning since the 1970s.

Neural networks with deep architecture have become the dominant approach for acoustic modeling in ASR, achieving performance levels comparable to human abilities in several ASR tasks. This success is largely attributed to the availability of large training datasets and advanced computational resources (Zhang, 2019). Despite this progress, developing fully autonomous ASR systems remains challenging due to the wide variability in human speech and accents, with these differences being even more pronounced in bilingual or multilingual speakers compared to those who are monolingual (Malik, 2021).

ASR and spoken language systems have made notable advancements in both technology and application. Nonetheless, certain technological limitations still hinder flexible solutions and user satisfaction. These challenges include sensitivity to environmental factors like background noise and insufficient integration of grammatical and semantic understanding (Benzeghiba, 2007).

Several studies have been published in recent years to evaluate and analyze various aspects of ASR models developed over time. A notable survey by Vadwala (2017) explored the challenges encountered by ASR systems and highlighted key ASR models. The report discusses several factors, such as utterance methods and styles, speaker variability, vocabulary size, and channel differences, among others. However, under-resourced languages receive little attention due to limitation in computational resources. Therefore, in the study the focus is on developing an E2E ASR system leveraging CTC techniques for the Tshivenda language.

2.2.1 Pronunciation modelling

Model pronunciation

Pronunciation modeling refers to the methods and frameworks used to generate high-quality, human-like pronunciations. This concept has been applied in voice recognition technology, as noted by Byrne et al. (1997). It has been demonstrated that pronunciation modeling improves the performance of ASR systems in various contexts. Linguistically, variations in pronunciation are often attributed to gaps in vocabulary, pronunciation, and another language-specific knowledge (Shivakuma, 2014). A speech recognition system's vocabulary includes pronunciation models that describe how words are articulated as

sequences of sub-word units, typically phonemes. By learning pronunciation rules from data rather than predefined pronunciation variants, pronunciation modeling can combine the strengths of both data-driven and rule-based approaches. Recognizing common linguistic errors also contributes to improving system performance through pronunciation modeling (Cremelie, 2014).

A speech recognition system typically trains on and decodes using a fixed set of pronunciations from the dictionary. Speech technologies, like ASR, need a large corpus of speech to collect and manually annotate, as well as a dictionary of every word that could possibly be used in the language with every possible pronunciation (Plauche, 2006). The dictionary would be too big to use if it tried to cover every potential pronunciation (Kanokphara, 2003). Possibly the most expensive component of development is the language resources, which include training data and pronunciation dictionaries. It is optional to provide a dictionary to limit the search field to words that are acceptable (Chan, 2016).

Pronunciation modelling relies on a variety of statistical and mathematical techniques to predict the pronunciation of words based on their phonetic and phonological properties. These models play a vital role in ASR systems by giving a structured way to handle the variability and complexity of spoken language. It uses statistical and mathematical models which some of them are illustrated below:

Hidden Markov Models (HMMs):

Hidden Markov Models (HMMs) are a crucial statistical tool in speech processing. An HMM is defined by a set of states, transition probabilities between these states, and emission probabilities that indicate the likelihood of observing a specific output given a particular state. The probability of a sequence of observed phonemes $O = (o_{1,}\, o_2, o_T)$ given a sequence of hidden states $Q = (q_{1,}\, q_2, q_T)$ is determined using the below formula:

$$P(O,Q) = P(q_1)\prod_{t=2}^{T} P(q_t|q_{t-1})\prod_{t=1}^{T} P(o_t|q_t) \quad (1)$$

HMMs are effective because they model speech as a sequence of discrete states (phonemes) and can capture temporal dynamics by defining transitions between these

states. This makes them suitable for handling sequential data like speech (Cremelie, 2014).

Gaussian Mixture Models (GMMs):
Gaussian Mixture Models (GMMs) are commonly used alongside Hidden Markov Models (HMMs) to model the distribution of acoustic features associated with each phonetic unit. A GMM represents the probability of an observation x given a state s as a weighted sum of multiple Gaussian distributions. The formula for this is:

$$P(x|s) = \sum_{k=1}^{K} \omega_k N(x|\mu_k, \sum_k) \quad (2)$$

where K is the number of Gaussian components in the mixture, ω_k is the weight of the $k-th$ Gaussian component with $\sum_{k=1}^{K} \omega_k = 1$, $N(x: \mu_k, \sum_k)$ represent a Gaussian distribution with μ_k and covariance $\sum_k$ (Cremelie, 2014). GMMs are effective in capturing the multimodal nature of speech features, providing a more accurate representation of the variability in pronunciation. By modeling the distribution of acoustic features with a mixture of multiple Gaussian distributions, GMMs can account for the diverse ways in which a phonetic unit might be realized, taking into consideration factors such as speaker variability, environmental noise, and contextual influences. This capability enhances the robustness and flexibility of speech recognition systems, enabling them to handle a wide range of pronunciations and acoustic conditions.

2.2.2 Acoustic Models

Artificial Neural Network

Artificial Neural Networks (ANNs), also known as Neural Networks (NNs), are computational frameworks modeled after the structure and functioning of biological neural networks found in the brain. These systems comprise artificial neurons interconnected nodes that simulate the behavior of biological neurons. Just like synapses in the brain, the connections between neurons transmit signals. When an artificial neuron receives input, it processes the information and sends a response to the neurons it is connected to. This process enables ANNs to learn and make decisions based on patterns in data, making

them highly effective for tasks like speech recognition, image processing, and natural language understanding.

ANNs are inspired by early models of sensory processing in the brain. By simulating networks of artificial neurons on a computer, ANNs can be trained to address problems like those processed by biological neurons. In this model, the connections between neurons, known as synapses, are represented by weights, which control the strength of input signals. A transfer function is applied to represent the nonlinear behavior of the neurons. The output of a neuron is determined by the weighted sum of the input signals, which is then transformed by this function (Abraham, 2005). This architecture enables ANNs to learn complex relationships and make predictions based on data.

A threshold unit is a specific type of model neuron. Its output is determined by whether the total weighted input exceeds a threshold value: the output is 1 if the threshold is surpassed, and 0 otherwise. When the total weighted sum matches the threshold, the output transitions from 0 to 1. This threshold behavior defines a hyperplane, representing a set of points in the input space that satisfy this condition (Krogh, 2008).

An artificial neuron's learning ability is realized by adjusting its weight using a defined learning algorithm (Abraham, 2005). To accurately solve a classification problem, particularly when the problem is separable, the weights and thresholds must be correctly calibrated. This is achieved through an iterative process where the network is presented with labeled samples one by one. This process, which mimics human learning, is called training or learning (Krogh, 2008).

In their article, Yang (2020) presented a primer on using Artificial Neural Networks (ANNs) for neuroscientists. ANNs offered a novel approach for modeling complex behaviors, diverse neuronal activity, and circuit connections, as well as exploring neural system optimization in ways that traditional models cannot achieve. The article demonstrated how ANNs have been effectively utilized in addressing neuroscientific challenges.

Focusing on aligning the mathematical framework of ANNs with neurobiology, the authors showed how modifications to the analysis, structure, and learning processes of ANNs can help address various brain research questions more effectively. The primer also included

tutorial-style code in PyTorch and Jupyter Notebook, providing practical examples and hands-on experience with the core concepts**.**

Deep Neural Network

A Deep Neural Network (DNN) is a type of ANN characterized by multiple hidden layers between the input and output layers. These networks are built using components such as neurons, synapses, weights, biases, and activation functions. Over recent years, DNNs have become a preferred choice for various computer vision tasks, including image classification, audio recognition, and natural language processing. In several cases, DNNs have achieved accuracy levels comparable to human performance (Montavon, 2018).

Modern DNNs exhibit intriguing behavior when trained on image data: their first-layer features often resemble Gabor filters or color blobs. This pattern is so prevalent that deviations from it in a natural image dataset may indicate problems with hyperparameters or software configurations. This phenomenon is consistently observed across different datasets and training objectives, such as supervised image classification (Krizhevsky et al., 2012), unsupervised density estimation (Lee et al., 2009), and unsupervised sparse representation learning (Lee et al., 2011) (Yosinski, 2014).

DNNs consist of sequentially arranged neurons, where each neuron processes inputs derived from the activations of neurons in the previous layer. Each neuron performs a simple computation, typically involving a nonlinear activation function followed by a weighted sum of inputs. Collectively, these neurons establish a complex nonlinear mapping between the input and output. This mapping is learned through the adjustment of neuron weights using a method known as error backpropagation (Rumelhart, 1986). In DNNs, the computations across layers are progressively integrated, with deeper layers capturing increasingly abstract features.

One study examined the application of DNNs in object detection (Szegedy, 2013). In this research, the authors made significant strides in the field by addressing the dual challenge of classifying objects and accurately localizing them within images. They introduced a simple yet effective method for framing object detection as a regression problem, utilizing

bounding box masks. Moreover, they proposed a multi-scale inference technique that allows several network applications to perform high-resolution object detection at a reduced computational cost. The method's effectiveness was demonstrated using the Pascal VOC dataset. The article concluded that combining a multi-scale, coarse-to-fine approach with the DNN-based object mask regression formulation resulted in impressive performance.

This technique is rarely employed due to its need for a large dataset to achieve better performance compared to other methods. The complexity of the data models makes the training process resource intensive. Furthermore, there is no established theory to assist in selecting the right deep learning tools, as it requires specialized knowledge in areas such as topology, training methods, and other specific factors.

Recurrent Neural Network

Recurrent Neural Networks (RNNs) are a class of artificial neural networks characterized by connections between nodes that form a temporal graph, allowing the network to evolve over time (Graves, 2010). Unlike conventional feedforward networks, RNNs can handle input sequences of varying lengths by leveraging an internal state, often referred to as memory. This allows RNNs to handle temporal data without requiring prior knowledge beyond the input and output representations. The internal state is crucial for modeling time series data (Graves, 2010). Furthermore, Bidirectional Recurrent Neural Networks (BRNNs) expand on RNNs by enabling access to both past and future information during training, which alleviates the limitation of relying only on past data (Schuster, 1997).

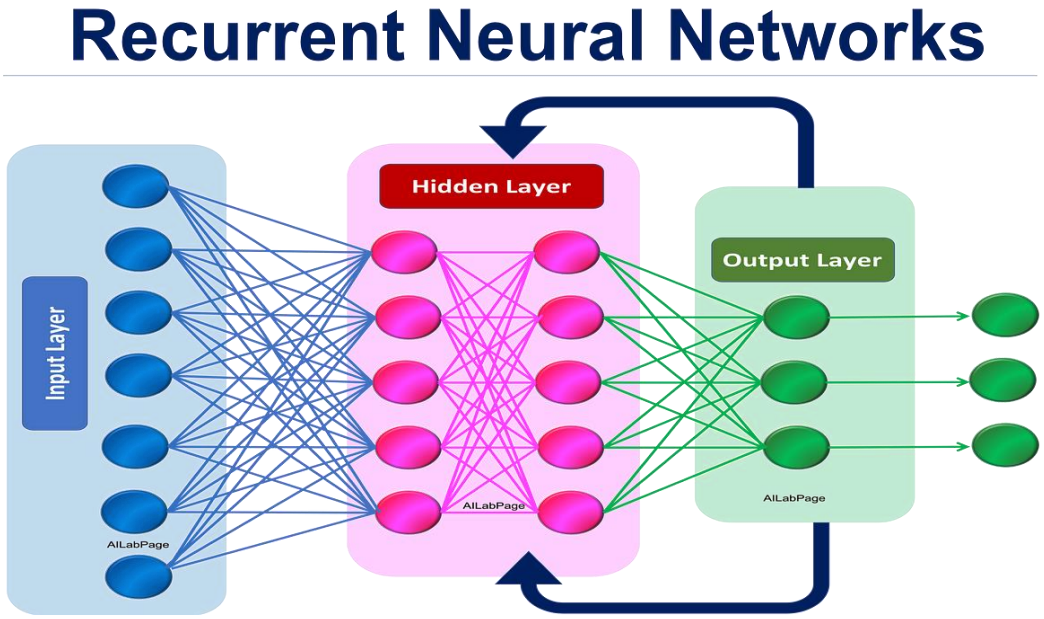

Figure 2.2 Recurrent Neural Network Architecture

Figure 2.2 illustrates the architecture of a Recurrent Neural Network (RNN). Its primary purpose is to visually represent the flow of information within an RNN, a type of neural network specifically designed for sequential data like speech. The diagram highlights the three main components of a basic RNN: the input layer, hidden layer, and output layer. Each of these layers plays a crucial role in processing sequential information. The input layer represents the input data, such as a sequence of audio features (e.g., spectrograms) in speech recognition. Multiple nodes signify that each time step might consist of multiple features. The hidden layer is the core of the RNN, where the output at a given time step depends on both the current input and the previous state. The looping arrow illustrates this recurrent connection, emphasizing how past information influences future predictions. Multiple nodes in the hidden layer allow the network to process different aspects of the sequence simultaneously.

The output layer produces the network's final prediction, which could be phonemes, words, or sentences in speech recognition. The multiple nodes in this layer may correspond to different possible outputs. The diagram also includes blue lines representing the weighted connections between layers. These weights are learned during training and determine how much influence each input has on the final prediction. While the diagram presents a simplified view, real-world RNNs used in speech recognition are

far more complex. Advanced architectures like Long Short-Term Memory networks (LSTMs) and Gated Recurrent Units (GRUs) are often incorporated to address challenges such as the vanishing gradient problem. The image effectively conveys the fundamental principle of sequential information processing, highlighting how RNNs capture temporal dependencies in speech.

Long Short-Term Memory

A Long Short-Term Memory (LSTM) network is a specialized type of RNN that enhances memory retention, making it particularly well-suited for capturing and learning long-term dependencies (Donges, 2021). The layers within an RNN that are commonly called LSTM networks are built using LSTM units. While both LSTM networks and their predecessor, the standard RNN, share similar underlying inference formulas, LSTMs are specifically engineered to handle long-duration learning more efficiently (Sherstinsky, 2020).

Sepp Hochreiter and Jürgen Schmidhuber, the inventors of LSTM in 1995, are regarded as the pioneers of this technology. LSTM units or blocks, which are integral parts of a recurrent neural network, are designed to utilize specific memory processes that enable AI systems to better mimic human reasoning. These specialized units excel in retaining information over both short and long periods. Their memory capacity is mainly attributed to the absence of an activation function in their recurrent components, which allows them to store and manage information effectively (Chen, 2017).

LSTM networks provide enhanced control over data retention and the selective discarding of past information, making them particularly effective for analyzing time series data, especially when significant events are separated by delays. They were specifically developed to address the vanishing gradient problem encountered during the training of traditional RNNs. Consequently, LSTMs often surpass RNNs, hidden Markov models, and other sequence-learning approaches in managing long-term dependencies within data. Training an RNN with LSTM units is a supervised process, where network weights are adjusted based on the error gradient calculated at the output layer. This optimization is typically performed using gradient descent, with backpropagation through time employed to compute the gradients needed for the process.

Convolutional Neural Network

Convolutional Neural Networks (CNNs) are a popular machine learning technique, particularly in computer vision, due to their powerful image classification capabilities. As a type of ANN, CNNs are specifically designed for processing visual data. They utilize a shared-weight architecture that employs convolutional filters or kernels, which move across input features to generate translation equivariant outputs, also known as feature maps. CNNs are sometimes referred to as Shift Invariant or Space Invariant Artificial Neural Networks (SIANN). These networks typically consist of several 2D convolution layers, which perform most of the computational work (Chang J, 2016).

CNNs are a specialized type of multilayer perceptron. While multilayer perceptrons are typically fully connected networks, where each neuron in one layer is linked to every neuron in the next, CNNs take a different approach to regularization by utilizing the hierarchical nature of data. Instead of connecting every neuron, CNNs use filters to capture simple patterns, which combine to form more complex ones. This approach allows CNNs to operate with less connectivity and complexity compared to fully connected networks, reducing the risk of overfitting. For instance, a 2D CNN model was used to extract local and global emotion-related features from voice and long-mel spectrograms (Zhao, 2019). This model focuses on analyzing spectral data by considering only the spatial correlation within the image's channels, offering similar performance with significantly reduced computational demand (Koundinya, 2018).

The core operation in a 2D Convolutional Neural Network (CNN) is the convolution, where a filter (or kernel) is applied to an input feature map to generate an output feature map. The mathematical expression for this operation, where a single filter K is applied to an input feature map I, is given by:

$$(I * K)(i,j) = \sum_m \sum_n I(i+m, j+n) * K(m,n) \quad (3)$$

where:

- $(I * K)(i,j)$ is the output feature map.
- $I(i+m, j+n)$ is a patch of the input feature map.
- $K(m,n)$ is the kernel or filter.

In pronunciation modeling, the input feature map I typically represents a 2D representation of speech, such as a Mel-spectrogram, which captures time-frequency information. The convolution operation helps in extracting local features from these representations, such as formants and harmonics, which are critical for accurate pronunciation modeling (Cremelie, 2014).

After performing a convolution operation in a 2D Convolutional Neural Network (CNN), a non-linear activation function is applied to improve the model's ability to capture more intricate patterns in the data. The Rectified Linear Unit (ReLU) is one of the most used activation functions in CNNs, and it is defined as:

$$ReLu(x) = \max(0, x) \tag{4}$$

ReLU is particularly beneficial for speech-related tasks because it addresses the vanishing gradient problem, which can impede the training of deep networks. By ensuring that gradients do not shrink excessively during backpropagation, ReLU helps maintain the flow of error signals throughout the network, allowing the model to learn more effectively and converge faster.

2.2.3 Language Models

Bigram Language

The bigram model estimates the probability of a word based on the preceding word, using conditional probabilities (Kapadia, 2019). To incorporate a look-ahead in a bigram language model, bigram probabilities are computed over the lexical tree nodes for each instance of the tree. It is anticipated that integrating the bigram language model with the beam search method will enhance its performance (Ortmanns, 1996).

Trigram Language

The trigram language model's probability can be modified to better match the present material to improve the language model. The probability distribution for the following word in the trigram language model is dependent on the preceding two words (Jelinek, 1991).

The actual set of trigram probabilities in each grammar is typically a small subset of the total number of trigram probabilities. For difficult vocabulary tasks, they typically number in the millions (Ravishankar, 1996).

2.3 End-to-End Architecture for Speech Recognition

ASR systems have evolved through significant advancements in deep learning, leading to the development of E2E architectures that simplify the traditional multi-stage pipeline of ASR. The two main categories of E2E designs, CTC and Attention-based mechanisms, represent distinct approaches for mapping audio sequences to text without requiring extensive preprocessing or alignment. Each method addresses specific challenges inherent in processing continuous speech, such as handling variable input lengths and managing dependencies across sequential data. By examining these designs, this chapter explores the foundations of CTC and Attention approaches and their roles in building robust, efficient, and scalable ASR systems.

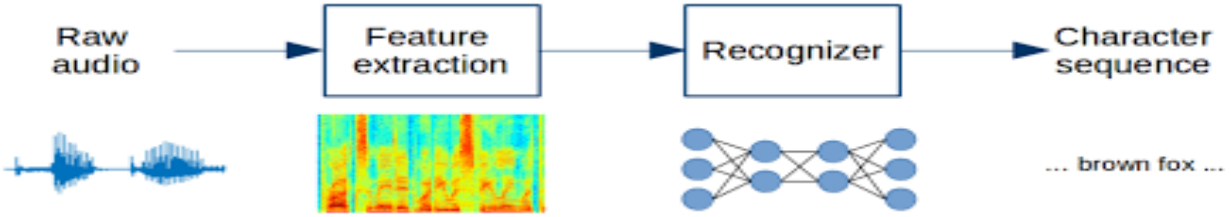

Figure 2.3 End-to-End Architecture

Figure 2.3 depicts a typical end-to-end architecture for automatic speech recognition (ASR), illustrating the sequential processing of audio data into text. The process begins with raw audio, visually represented as a waveform. This raw sound data is unprocessed and not directly suitable for recognition. To prepare it for analysis, the feature extraction stage converts the waveform into a more informative representation, such as a spectrogram, which captures frequency variations over time. Common feature extraction methods include Mel-Frequency Cepstral Coefficients (MFCCs) and filter banks, both of which help highlight important speech characteristics, making it easier for the recognizer to process the data.

The recognizer is the core of the ASR system, typically depicted as a neural network that maps the extracted features to a sequence of characters or words. While the image

illustrates a fully connected network, other architectures like Recurrent Neural Networks (RNNs) or Convolutional Neural Networks (CNNs) can also be used. The complexity of this network determines how well it handles variations in accents, and different speaking styles. Finally, the character sequence represents the transcribed. This end-to-end pipeline—from raw audio to text—functions as a single, integrated system, ensuring a streamlined approach to speech.

2.3.1 Connectionist Temporal Classification

Connectionist Temporal Classification (CTC) is a term that refers to the outputs and scoring of a neural network and was first used in 2006. The CTC is a remarkable solution for avoiding pre-segmentation of training examples and post-processing to convert a RNN's output into label sequence. The CTC method enables neural networks to be trained with pairs of input data and corresponding target labels (text). The neural network is programmed to output labelling in a particular coding scheme (Scheidl, 2018). When utilizing the CTC approach, a new unit is established to represent the likelihood of a blank label (i.e., no class label) (Soullard, 2019). The CTC loss function facilitates end-to-end neural network training for sequence-to-sequence tasks without requiring pre-aligned input and output sequences. Traditionally, CTC has been used to train sequential, single-label problems, where each element in the sequence corresponds to only one class (Chen, 2017).

A quick method is needed to compute the conditional probability of individual labeling, p(l | x). According to Salazar (2019), this can be difficult as it requires summing over all possible paths corresponding to a given labeling, which are numerous. However, a dynamic programming approach, such as the forward-backward algorithm used for HMMs, can address this challenge (Rabiner, 1989). The main concept is that the sum over paths for a given labeling can be divided into an iterative sum over paths that align with the prefixes of the labeling. By using recursive forward and backward variables, the computations can be done efficiently.

We examine a modified label sequence l, where blanks are inserted at the beginning, end, and between each pair of labels to allow for blanks in the output paths. This modification

leads to a sequence length of 2|I| + 1. When calculating the probabilities for the prefixes of the sequence, transitions are allowed between blank and non-blank labels. Any prefix can either begin with a blank (b) or the first symbol in the label sequence (I1).

In CTC systems, letters or words can serve as output labels, with blank labels typically occurring more frequently. Non-blank labels often show posterior spikes. A simple approach to generating ASR outputs with CTC is to concatenate the non-blank labels associated with these posterior spikes and combine them into word outputs when necessary (Li, 2019). This approach, known as greedy decoding, is particularly appealing for E2E models because it eliminates the need for a language model (LM) or complex decoding strategies. The simplicity of greedy decoding aligns with the notion that a language model is not required in this context.

The CTC used in non-autoregressive E2E ASR with Mask Predict allows for faster inference by generating outputs in parallel, rather than sequentially, as noted by Higuchi (2020). In their tests, three tasks with different languages and amounts of training data were used: the 81-hour Wall Street Journal (WSJ) dataset in English, the 581-hour corpus of spontaneous Japanese (CSJ), and the 16-hour Voforge dataset in Italian. The network inputs included 80 mel-scale filter bank coefficients along with three-dimensional pitch characteristics, and the model was trained using SpecAugment. For tokenization, characters were utilized: Latin alphabets for English and Italian, and Japanese syllabic characters (Kana) and Chinese characters (Kanji) for Japanese. The findings showed that, when compared to conventional CTC models, the Mask CTC with the mask-predict objective delivered better results than the greedy CTC outputs for non-autoregressive models. Additionally, the model's performance was further enhanced through refinement using the proposed CTC masking technique.

The performance was further enhanced by increasing the number of decoding iterations, with mask iterations yielding the best results. When compared to previous studies, the results from Mask CTC were found to be reasonable. Mask CTC surpassed the standard CTC model, achieving results comparable to those on the Wall Street Journal dataset, suggesting that this approach can be applied to different languages with a moderate amount of training data. Although Mask CTC resulted in lower Character Error Rates

(CERs) than the autoregressive model, the improvements over the basic CTC model were modest across the tasks. The experiments concluded that Mask CTC outperformed the regular CTC model while maintaining high decoding speed. Looking forward, the team plans to bridge the gap between random masking during training and CTC output inference.

CTC is used in the architecture for E2E speech recognition. In their experiments, the researchers employed the WSJ1 and WSJ0 clean speech corpora, alongside the CHiME-4 noisy speech corpus. They also included the HKUST Mandarin Chinese and Corpus of Spontaneous Japanese (CSJ) conversational telephone speech to diversify the datasets. The findings revealed that the hybrid CTC architecture outperformed attention based E2E ASR by effectively addressing misalignment issues. Unlike traditional Japanese and Mandarin Chinese ASR systems, this approach does not depend on linguistic resources such as a morphological analyzer, pronunciation dictionary, or language model.

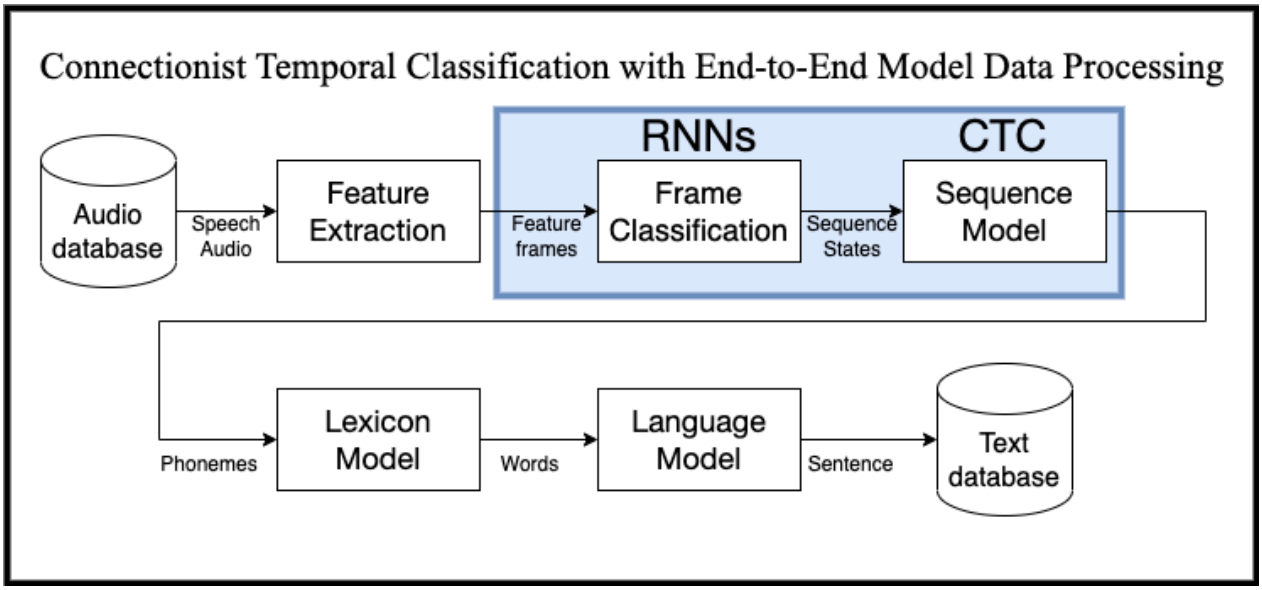

Figure 2.4 Connectionist Temporal Classification Architecture

Figure 2.4 illustrates the architecture of a speech recognition system using CTC, detailing each stage of the process. The system begins with data input, which consists of an audio database containing numerous speech recordings used for training and testing. Next, the feature extraction stage processes the raw audio to derive meaningful acoustic features, such as Mel-Frequency Cepstral Coefficients (MFCCs) or spectrograms. These features

provide a compact yet informative representation of the audio signal, discarding irrelevant information while preserving essential speech characteristics.

The core of the model is the Recurrent Neural Network (RNN), which performs frame classification by analyzing short audio segments and mapping them to a probability distribution over phonemes. Because RNNs retain temporal dependencies, they can capture the sequential nature of speech. The sequence states output from this stage represent phoneme probabilities across all frames. The CTC layer then aligns the RNN's output with the actual phoneme or word sequence, handling variations in input and output lengths. It allows for blank tokens to accommodate pauses and insertions, ensuring flexible alignment. Optional language and lexicon models further refine the transcription by enforcing grammatical correctness and mapping phonemes to words. Finally, the system produces a transcribed text output, stored in a text database. Overall, the diagram presents an end-to-end speech recognition pipeline that efficiently processes raw audio into meaningful text.

CTC is a training algorithm designed for sequence-to-sequence tasks where the alignment between input and output sequences is unknown. It is particularly beneficial in speech recognition and handwriting recognition applications. CTC enables neural networks to learn probabilistic alignments between inputs and outputs, facilitating the modeling of sequential data without explicit alignment annotations.

Mathematically, CTC modifies the standard neural network architecture by introducing a special 'blank' token in the output layer. This allows the network to output a probability distribution over possible labels, including the blank, at each timestep. The CTC loss function computes the negative log probability of all possible label sequences that could align with the given input sequence, summing over all possible alignments. This approach enables the training of neural networks on unsegmented sequence data.

For a comprehensive visual and mathematical understanding of CTC, refer to the guide by Hannun et al. (2017), which provides an in-depth exploration of the algorithm's mechanics and applications.

Future research will focus on applying this method to other languages, including English, with particular attention to overcoming challenges associated with long sequence lengths, which result in high computational costs and complicate the training of decoder networks. The reason for choosing this algorithm is that CTC is a neural network output designed to address sequence problems, such as handwriting and speech recognition, where the timing of the sequence can vary. CTC serves as a scoring function used to train RNNs, including LSTM networks, for tasks that involve sequences with fluctuating timings. By using CTC, there is no need for a pre-aligned dataset, simplifying the training process significantly.

2.3.2 Attention

Attention is a sequence-to-sequence model frequently used for tasks in natural language processing (Cui, 2018). The attention mechanism enables the decoder to concentrate on specific parts of the input sequence, enhancing the alignment between auditory frames and recognized symbols. By utilizing an encoder-decoder architecture, the attention-based E2E model addresses the automatic speech recognition (ASR) challenge by mapping speech feature sequences to text (Watanabe, 2017). In this framework, the attention mechanism decides which encoder features should be given priority (Miao, 2017). When decoded using independently trained language models, attention-based E2E systems typically achieve improved recognition performance.

The attention-based approach directly computes the posterior, p(C|X), by applying a probabilistic chain rule, without assuming any conditional independence between the variables (Watanabe, 2017).

2.4 Tshivenda Language

We chose Tshivenda as our language of research since it is one of the official and under-resourced languages of the Limpopo province, hence we want to try the CTC approach on it and produce working speech processing resources.

Tshivenda language is today spoken by approximately 875 000 people in South Africa, predominantly in the Limpopo Province (Census, 2021). The Venda’s history dates to the Mapungubwe Kingdom (9th Century). The Vhaluvhu of Hamashamba and Mulima

dominate the Venda clans who speak the Tshiguvhu dialect in the southwestern part of the country. The Venda tribe is made of twenty-one clans, with the Vhaluvhu being one of them (Mulaudzi, 1996). The writing system of the Tshivenda language's alphabet is based on five accented characters in the Latin alphabet. 5 special characters (Ṅ, Ḓ, Ḽ, Ṋ and Ṱ) e.g. ḓuvha, ṱhoho, ḽivhuwa, ṋamani etc. are used in dentistry that has a circumflex accent below the letter and an over the dot for velar. Seven vowels are written using five vowel letters. Only foreign words and names are written with the letters C, J and Q (Poulos, 1990).

In Tshivenda, special diacritical marks are used on certain Latin letters to represent distinct sounds not found in the standard Latin alphabet. These marked letters help accurately convey sounds essential to the language. Here's a breakdown of each letter and its significance:

- Ṅ (N with dot above): This represents a nasalized sound, which is produced by allowing air to escape through the nose. In Tshivenda, it emphasizes a nasal quality and is distinct from the regular "N" sound. It might sound closer to "ng" in English, but with the dot, it has a specific local resonance in words.
- Ḓ (D with circumflex below): The circumflex below the "D" indicates a dental sound, meaning it is produced by placing the tongue against the upper teeth. This differs from the regular "D" sound, giving it a softer pronunciation, typical in words like ḓuvha (meaning "day").
- Ḽ (L with circumflex below): Like Ḓ, this "L" with a circumflex below represents a dentalized version of the sound. The pronunciation requires touching the tongue to the upper teeth rather than the alveolar ridge (the part of the mouth just behind the teeth), making it unique in Tshivenda.
- Ṋ (N with circumflex below): This is also a dental sound. The circumflex indicates that, like Ḓ and Ḽ, the tongue should touch the upper teeth when pronouncing it. This makes it sound distinct from a regular "N," giving it a softer, more muted sound in the language.
- Ṱ (T with circumflex below): This represents a dental version of "T," where the tongue touches the teeth rather than the alveolar ridge. This letter is pronounced

softer and with a different tone than a standard "T" sound, found in words such as ṱhoho (meaning "head").

2.5 Conclusion

In this chapter, we explored the foundational E2E architecture that have transformed ASR by simplifying and streamlining the traditional ASR pipeline. Specifically, we examined the CTC and Attention-based approaches, each offering unique strengths for handling the complexities of speech-to-text conversion. The CTC approach provides a robust framework for sequence-based tasks, eliminating the need for prior alignment, which makes it especially suited for ASR applications that deal with continuous speech input of variable lengths. The incorporation of blank labels and dynamic programming enhances CTC's flexibility, enabling it to handle the challenging alignment and timing inconsistencies that are common in spoken language. On the other hand, the attention-based mechanism, which uses encoder-decoder architecture, improves recognition accuracy by allowing the model to focus on relevant segments of the input sequence. This approach effectively addresses misalignment issues, especially in languages with intricate grammatical structures and pronunciation variations.

The Tshivenda language, as an under-resourced language, presents a unique application for these ASR methodologies. Given its limited digital resources, the simplicity and efficiency of the CTC approach hold significant promise for developing robust, scalable ASR systems that do not rely heavily on extensive linguistic resources, such as language models or pronunciation dictionaries. By adapting and refining these end-to-end methods, this research aims to contribute valuable tools and resources for Tshivenda, helping to bridge the technological gap in ASR for South African indigenous languages.

Chapter 3: Methodology

3.1 Introduction

In this chapter, we get into the details of the corpus, its characteristics, and its role in our research. The National Centre for Human Language Technology (NCHLT) speech corpus serves as the foundation for our investigation. Our investigation delves into various dimensions of this corpus, including the distribution of speakers, words, sentences, and duration across different sets. Furthermore, our study encompasses data visualization techniques.

We present graphical representations of words' frequencies within the training, testing, and validation sets. We delve into loading the speech dataset, preprocessing steps, creating dataset objects, and visualization of the data. Finally, our evaluation phase encompasses metrics such as the CTC loss and WER. We explain how these metrics help us assess the model's accuracy and alignment with the actual transcriptions, ensuring a comprehensive understanding of its performance.

3.2 Tshivenda Speech Corpus

The application of CTC to the Tshivenda language presents unique challenges and opportunities. Tshivenda, a Bantu language spoken in South Africa, has a rich phonetic inventory and complex morphological structures. The limited availability of large, annotated speech corpora for Tshivenda necessitates the use of techniques that can effectively handle unaligned and sparse data.

CTC's ability to align input and output sequences probabilistically makes it a suitable choice for modeling the phonetic nuances of Tshivenda. By leveraging multilingual training strategies and cross-lingual adaptation techniques, as explored by Li et al. (2019), it is possible to enhance the performance of CTC-based models in low-resource languages.

The NCHLT speech corpus was used in this study. This dataset can be freely accessed using the link. It contains wide-band speech from 200 people in each of South Africa’s eleven official languages (Bernard, 2014). Tshivenda language is the selected language

among eleven official languages. A broadband speech corpus of roughly 56 hours that has been orthographically transcribed, with a test set of 8 speakers. The Tshivenda speech corpus has about 50 hours of orthographically transcribed speech in Tshivenda (Bernard, 2014).

The dataset was downloaded from the South African Centre for Digital Language Resources (SADiLaR). It includes two main folders containing audio files. Within these folders, there are subfolders with audio files in ".wav" format and corresponding transcriptions stored in XML files.

The training metadata was divided into a training set and a validation set. The training set comprised 90% of the training metadata, while the validation set accounted for the remaining 10%. The entire testing metadata was used for the testing set.

Table 3.1 Summary of the clean Tshivenda corpora

Speakers		Words		Sentences	Duration(hrs)
Males	Females	Types	Tokens	49 750	56:19
125	83	7150	25 557 675		

In Table 3.1, we show that the stratum consists of 208 speakers of which 83 are female speakers and 125 are male speakers intervening speech (3–5-word utterances read from a smartphone screen). The speech corpus has a total of 7150 words, and they appear 280853 times with a duration of 56 minutes and 19 seconds.

Table 3.2 Training and testing set

	Training set	Testing set	Validation set
Number of Sentences	42249	2806	4695
Number of Words	7428	3580	3335
Duration (hrs)	47:50	3:11	5:18

In Table 3.2, we show that we have a total of 49750 sentences used in the whole research where we split them into a training set which has 42249 sentences, testing set with 2806

sentences and validation set with 4695 sentences. We then split the sentences into words the way they are separated. The training set has 7428 words, the testing set has 3580 words, and the validation set has 3335 words. The training set has a duration of 47 minutes and 50 seconds, the testing set has a duration of 3 minutes and 11 seconds, and the validation set has a duration of 5 minutes and 18 seconds.

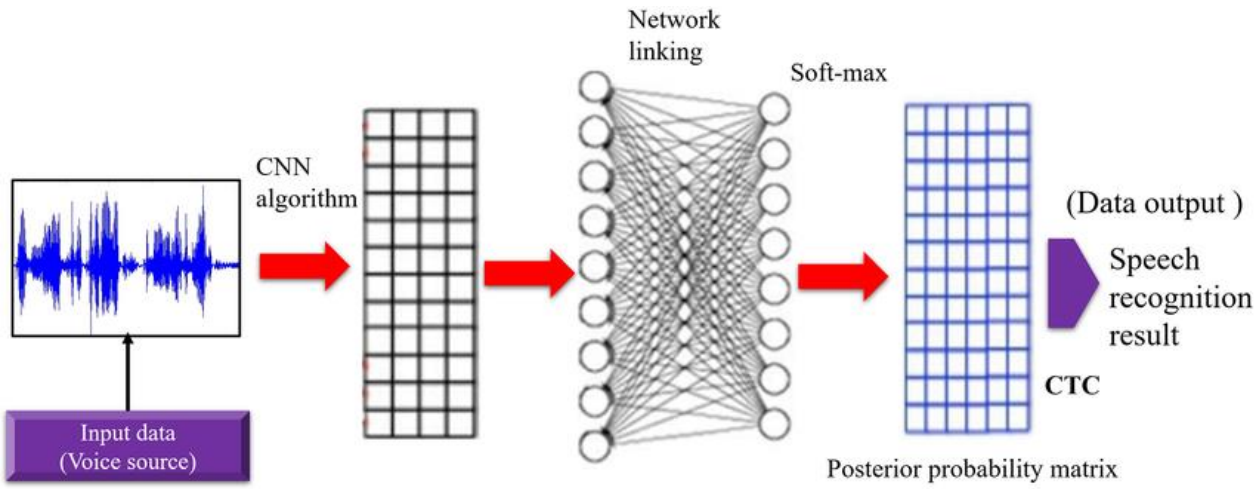

Figure 3.1 Training Flowing Chart

Figure 3.1 illustrates the process of training a speech corpus using CTC. The process begins with input data – a voice source represented as a spectrogram (a visual representation of sound frequencies over time). This is the raw audio data that needs to be transcribed. This input is then fed into a Convolutional Neural Network (CNN). CNNs are particularly good at processing grid-like data like spectrograms, extracting relevant features from the audio signal. The output of the CNN is a feature representation of the audio.

The feature representation from the CNN is then passed to a Recurrent Neural Network (RNN) (although the diagram doesn't explicitly label it as such, the structure of the network strongly suggests this). RNNs excel at processing sequential data like speech, capturing temporal dependencies within the audio. The network linking represents the connections between the neurons in this RNN, which learn to map the features to phonetic units. A softmax layer follows the RNN. The softmax function converts the raw outputs of the RNN into probability distributions over the possible phonemes (or characters) in the Tshivenda language alphabet. Essentially, for each time step in the audio, the softmax layer provides a probability for each phoneme. The output of the softmax layer is a matrix representing

the posterior probabilities of each phoneme at each time step. This is the key output for the CTC algorithm.

The Connectionist Temporal Classification (CTC) algorithm takes this posterior probability matrix as input. CTC is crucial because it handles the variable-length problem of speech recognition. Speech doesn't have a fixed number of phonemes per unit of time, and CTC resolves this by aligning the probabilities with the actual sequence of phonemes in the target Tshivenda text. It does this by finding the most likely sequence of phonemes that maximizes the overall probability given the input matrix, considering all possible alignments of the model output to the target Tshivenda transcriptions. The final output is the speech recognition result generated by CTC. This is the Tshivenda transcription derived from the audio input.

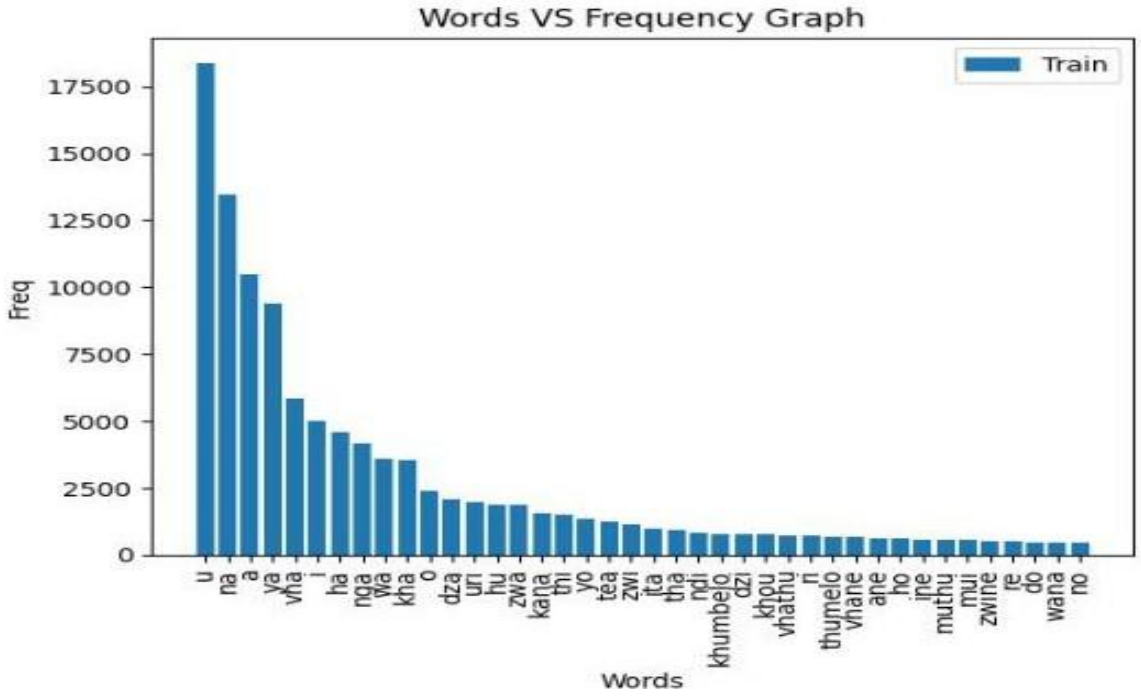

Figure 3.2 Words Frequency graph for Training Set

Figure 3.2 for the training set shows the top 40 words with their frequency for Tshivenda language as they appear in the NCHLT speech corpus. The word "*u*" is the most occurring word with 18304 occurrences in the training set. The word "*u*" has more occurrences because it is used to link verbs and the person/entity/thing that is taking the action that is being prescribed by the verb. It is also being used to concatenate two different sub-sentences to make one meaningful sentence. The word "rubriki" translation in english is

“rubric” which is a loanword, appears to be the smallest word with 1 count in the training set. It appears the least because they don’t usually use the word in most of the sentences.

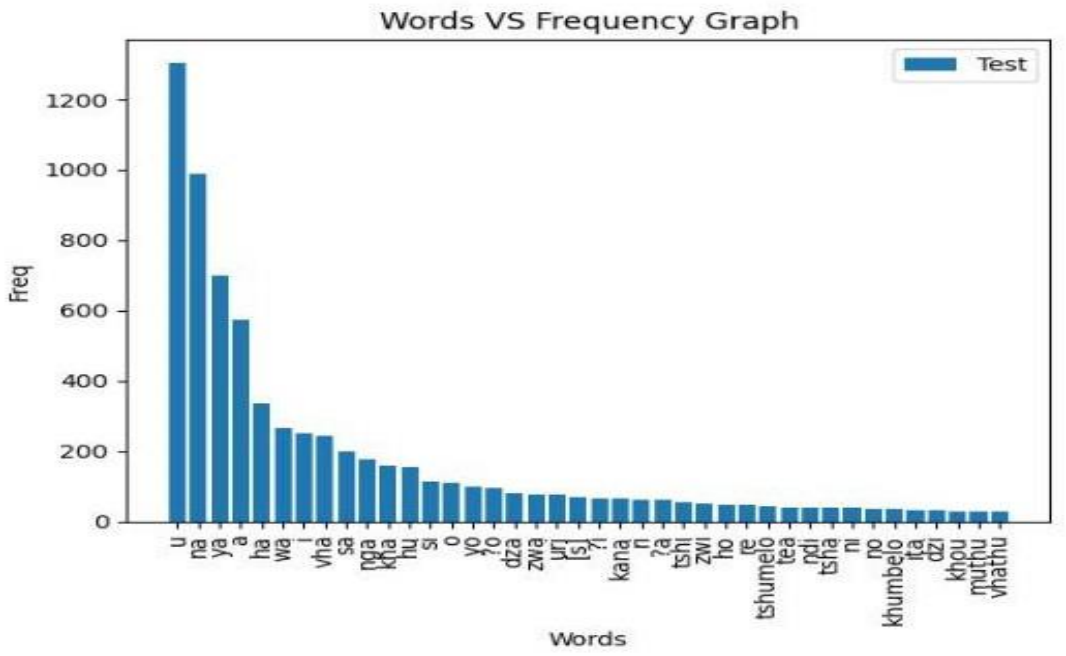

Figure 3.3 Words vs Frequency Graph Test Set

Figure 3.3 for the test set shows the top 40 words with their frequency for Tshivenda language as they appear in the NCHLT speech corpus. It is clear “u” is the most occurring word with 2082 occurrences in the testing set. The word “tshitalula” which translates in english is “introduction” appears to be the smallest word with 1 count in the training set. It appears the least because they don’t usually use the word in most of the sentences.

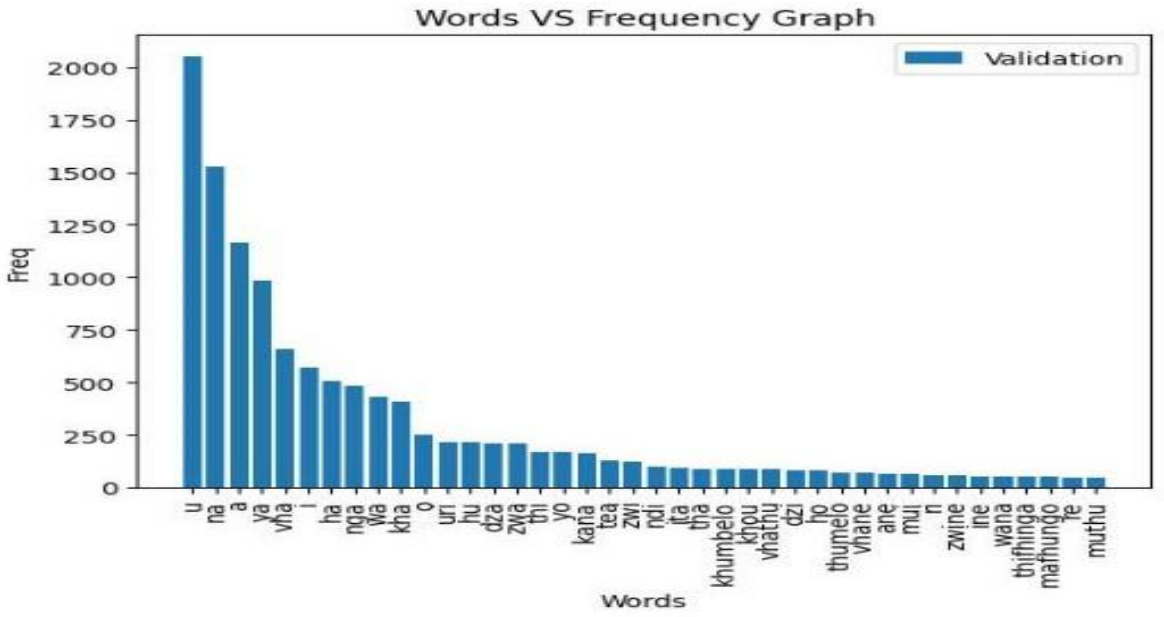

Figure 3.4 Words vs Frequency Graph Validation Set

Figure 3.4 shows the top 40 words with their frequency for Tshivenda language as they appear in the NCHLT speech corpus. It is clear that the word “u” is the most occurring word with 1304 occurrences in the validation set. The same pattern was observed for the test set for the word “u”. The word “isendekaho” which translates in english is “reliability” appears to be the smallest word with 1 count in the training set. It appears the least because they don’t usually use the word in most of the sentences.

3.3 Model Development

3.3.1 Tshivenda Speech Dataset

The audios were moved from multiple subfolders into the main folder. The metadata xml files were then changed to csv file, and both the metadata files and the audio file were uploaded on the google drive and accessed through data path. The corresponding csv files and audios were loaded to the notebook, and the csv was read as a data frame with the format [“file name”, “transcription”]. File name is the name of the corresponding .wave file, transcription is the words spoken by the reader (UTF-8).

3.3.2 Text Data Preprocessing

We assigned TensorFlow to utilize string manipulation methods, regular expression, or library functions to swap out the special characters for integers because the model can't compute the special characters, that is the reason we converted.

Then after we begin by preparing the vocabulary that will be used by the transcription's accepted character set, mapping character to integers and mapping integers back to original characters. The next step we created is the function that describes the transformation we will be applying to each element of our dataset, and we achieved that with an integer scalar tensor with the window length in samples.

3.3.3 Tshivenda Dataset Objects

We created a tf.data.Dataset object that produces the altered components in the same order as the input. It uses the tensor slices method to create a dataset from the input data containing file names and normalized transcriptions. Then, it applies the encoded single sample mapping function to each sample in the dataset, which is a custom function. After that, the dataset is padded to ensure that each batch has the same shape and size. Finally, the prefetch method is used to optimize performance by buffering data for the upcoming iterations.

3.3.4 Visualization of the Speech Data

Our dataset was visualized, together with the audio sample, spectrogram, and related label. We created a figure using "plt.figure" and iterated through the first batch of the dataset. It extracts the spectrogram, trims zeros from it, and assigns it to the spectrogram variable. We converted the label from a numerical representation to characters using the "number to character" function. The spectrogram is then plotted using "ax.imshow", and the title is set to the label. The axis is turned off for the spectrogram plot.

Next, the corresponding audio file is read and decoded using "tf.audio.decode_wav", and the waveform is plotted using "plt.plot". Then we set the title of the waveform plot to "Signal Wave", and the x-axis limits are adjusted to match the length of the audio. Finally, the

audio is displayed using “display.Audio”. The resulting figure, with both the spectrogram and the waveform, is shown using “plt.show()”.

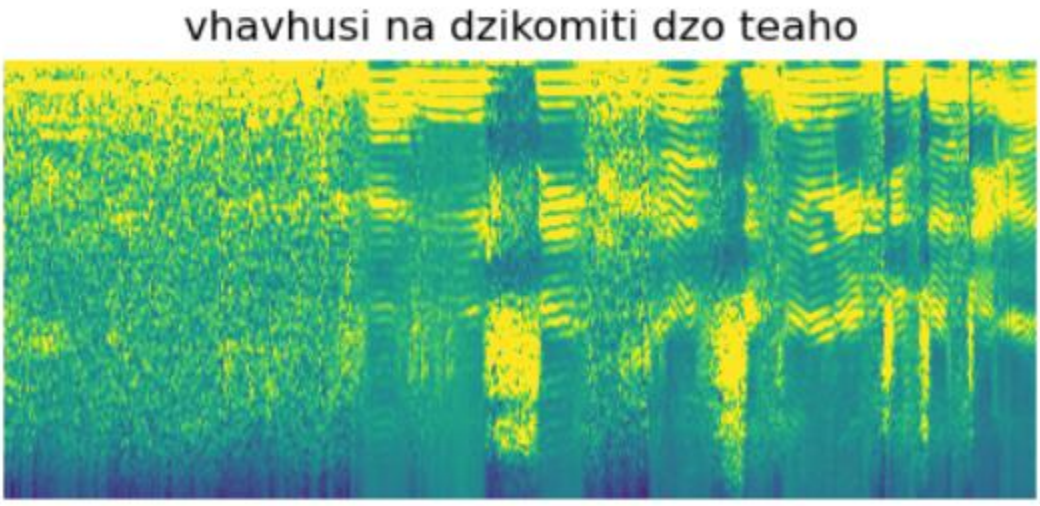

Figure 3.5 Spectrogram

Figure 3.5 shows the spectrogram for the sentence “vhavhusi na dzikomiti dzo teaho” where in English it means “Rulers and associated committees”. A spectrogram is a visual representation that shows how the frequency content of an audio signal changes over time. It demonstrates how various frequencies' energies shift over the course of the recording. The X-axis of the spectrogram shows time as it moves from left to right. Each point on the X-axis represents a distinct time interval in the audio example. The Y-axis represents the audio signal's various frequencies, which also shows frequency components. Lower places on the Y-axis correspond to lower frequencies, while higher positions to higher frequencies. The yellow color in the spectrogram indicates higher energy at a specific frequency and time, while the greenish color represents the lower energy level.

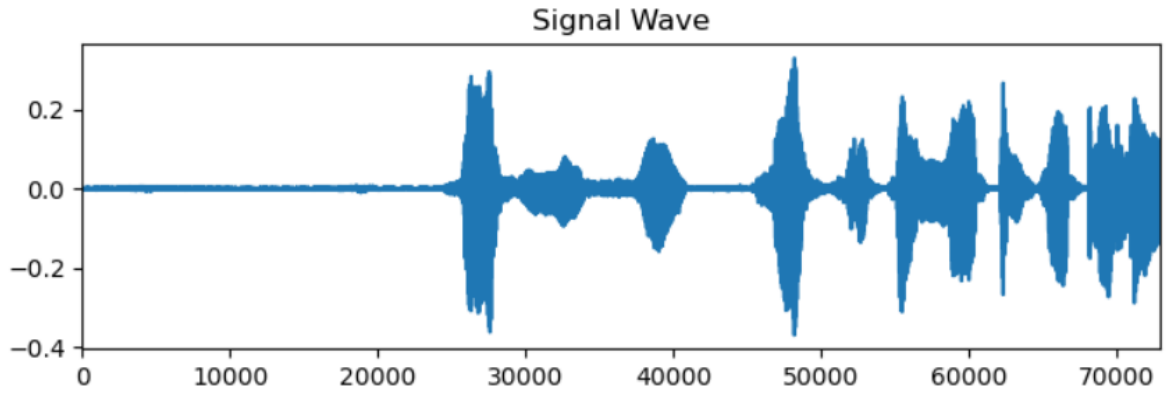

Figure 3.6 Signal Wave

Figure 3.6 shows the signal wave for the sentence “vhavhusi na dzikomiti dzo teaho”. An audio signal wave is a visual representation of how an audio signal's amplitude (loudness) fluctuates over time. It is a time-domain graph that shows how the strength of the sound changes as the audio goes on. The signal wave's X-axis shows time as it moves from left to right. Each dot on the X-axis represents a distinct time interval in the audio example. The Y-axis shows the audio signal's amplitude or intensity. The strength or loudness of the audio at any one time is represented by the height of the wave at that location on the Y-axis. The waveform's peaks, where the amplitude is at its highest level, correspond to loud sounds. These peaks may correspond to notable auditory elements or occurrences, such as the speaker's raised voice when speaking. The waveform's troughs correspond to the audio's quieter or softer tones. The amplitude values of these locations are lower. Sections of the waveform that are flat denote quiet or almost silence, where the amplitude of the audio stream is very low.

3.3.5 Connectionist Temporal Classification – based Model Training

First, we establish the CTC loss function. We determined the lengths of the input and label sequences by obtaining the shapes of “y_true” and “y_pred”. Here, “y_true” represents the true or ground truth labels of our dataset, while ‘Y_pred” represents the predicted labels generated by the model. To get their shapes, we used `tf.shape`. Calling `tf.shape(y_pred)` returns the shape of “‘y_pred” as a tensor.

The shape of “y_pred” typically includes the batch size and the length or dimensions of the predicted labels. “*tf.shape(y_pred)[1]*” extracts the length of the predicted labels from

the shape tensor. The obtained lengths are then cast to the int64 data type. Next, we created tensors “input length” and “label length” with the same length as the batch (batch length). These tensors are created using “tf.ones” to have a shape of (batch length, 1). This step is done to ensure that the “input length” and “label length” tensors have the same shape as the batch, which is required by the subsequent “ctc batch cost” function. Then, we computed the CTC loss using “keras.backend.ctc_batch_cost” by passing in “y_true”, “y_pred”, “input length”, and “label length”. The computed loss is then returned as the output of the function.

Here, we specify our model. We created a model that is similar to DeepSpeech2. The model architecture consists of convolutional layers, bidirectional GRU (recurrent) layers, a dense layer, and a classification layer. The model takes spectrogram inputs and predicts the corresponding labels. The input spectrogram is first reshaped to have an additional dimension to be compatible with 2D convolutional layers. Two convolutional layers with batch normalization and ReLU activation are applied to extract features. The resulting volume is then reshaped to match the expected input shape of the RNN layers.

The RNN layers, specified by the rnn_layers parameter, are created using GRU cells and wrapped in bidirectional layers for better performance. Dropout layers are added between each pair of bidirectional layers, except for the last one. This helps to prevent overfitting during training. After the RNN layers, a dense layer with ReLU activation and dropout regularization is applied. The final classification layer uses a SoftMax activation and predicts the output classes. The model is compiled with an Adam optimizer and the “CTCLoss” function as the loss function. Finally, the model summary is printed using “model.summary()” with a specified line length of 110.

We used the function “decode batch predictions” that takes the predictions as input and performs decoding to obtain the corresponding text outputs. The function first creates an array input length of the same length as the batch size, where each element is set to the length of the predictions. This is necessary to provide the input length information to the CTC decoding process. Then, the function uses the CTC decoding function “keras.backend.ctc_decode” with the predictions and input length to obtain the decoded results.

The “greedy=True” argument specifies that a greedy search strategy should be used for decoding. Alternatively, beam search can be used for more complex tasks. The decoded results are stored in the results variable. The function iterates over these results and converts them from numerical representations to characters using the “number to character” function. The resulting text is appended to the output text list. Finally, the function returns the output text list containing the decoded text for each prediction in the batch.

The callback class, which inherits from the “keras.callbacks.Callback” class, was used during training to display a few transcriptions after each epoch. The “**init**” method initializes the callback and accepts a dataset parameter, representing the dataset used for evaluation. The “on_epoch_end” method is triggered at the end of each training epoch. Within this method, the callback processes the dataset to generate predictions and targets. It iterates through each batch in the dataset, makes predictions using the model (model.predict(X)), decodes the batch predictions, and appends both predictions and targets to the respective lists.

After processing the dataset, the callback computes the WER between the targets and predictions using the “wer” function. It then prints a separator line and the calculated WER score. Following that, it randomly selects two indices from the predictions and displays the corresponding target and prediction pairs.

The fit function trains the model for the specified number of epochs, performing forward and backward passes through the network, optimizing the model's parameters, and evaluating its performance on the validation dataset at the end of each epoch. The validation callback is triggered at the end of each epoch to display transcriptions and assess performance. The training history, containing metrics and loss values, is stored in the history variable for future analysis or visualization.

3.3.6 Inference

We assessed the model's predictions on additional samples from the validation dataset by calculating the WER between the predicted transcriptions (predictions) and the actual transcriptions (targets). The code iterates through each batch in the validation dataset. For each batch, it retrieves the input data (X) and true labels (y), then makes predictions using the model (model.predict(X)). The predictions for the batch are decoded using the “decode batch predictions” function and added to the predictions list.

The true labels are also processed by converting them from numerical representations to characters using the “number to character” function. The resulting labels are appended to the targets list. After processing all the batches, the code calculates the WER between the targets and predictions using the “wer” function. The WER score is then printed. Finally, the code randomly selects five indices from the predictions list using “np.random.randint” and displays the corresponding target and prediction pairs.

3.4 Evaluation

The primary metrics used to assess the model's performance are WER and CTC loss. WER quantifies the number of errors in the predicted text compared to the target text, normalized by the total number of words in the target. A lower WER signifies better model performance. CTC loss, in contrast, measures how effectively the model is learning from the training data, with a lower loss generally indicating more efficient learning.

3.4.1 Connectionist Temporal Classification Loss

CTC loss is essential in training sequence-to-sequence models for tasks such as speech recognition. It allows the model to learn to predict sequences where the alignment between input and output is not one-to-one. The CTC loss manages variable-length outputs by incorporating "blank" tokens and using dynamic programming to compute the likelihood of potential alignments between the input and output sequences.

The CTC loss is implicitly used when training the model with the “model.fit” function. The CTC loss is computed internally during training to guide the optimization process. This

loss encourages the model to produce sequences of characters that match the target sequences.

CTC loss is used to align variable-length input sequences (audio waveforms) with variable-length output sequences (words). A CTC loss function used in training neural networks for sequence-to-sequence tasks. The formula for CTC loss can be expressed as:

$$L(X, Y) = -\log\left(P(Y|X)\right) \quad (5)$$

Here's a more detailed breakdown of the components:

- $L(X, Y)$ is the CTC loss between the input sequence X and the target sequence Y.
- $P(Y|X)$ is the probability of the target sequence Y given the input sequence X, computed by the neural network.
- The negative logarithm $-\log(*)$ is applied to the probability to measure how well the predicted sequence aligns with the target sequence. The closer the predicted sequence is to the target sequence, the lower the loss.

3.4.2 Word Error Rate

Word Error Rate (WER) is a metric used to assess the accuracy of transcriptions in speech recognition tasks. It measures the differences between the predicted and target transcriptions in terms of insertions, deletions, and substitutions required to convert one into the other. We calculated WER using the "wer" function. This involves comparing the predicted transcriptions (decoded using "decode_batch_predictions") with the actual target transcriptions (converted using "num_to_char"). The WER score is printed at the end of each epoch through the "CallbackEval callback."

"decode_batch_predictions" function decodes the output predictions of the model using CTC decoding. It converts the model's predicted probabilities into text using a greedy search approach. "CallbackEval" Class is the custom callback class that calculates the WER at the end of each epoch and prints it. It does so by comparing the decoded predicted transcriptions with the target transcriptions for the validation dataset. Additionally, it prints out a few randomly selected targets and predicted transcriptions to provide insight into the model's performance.

The formula for calculating WER involves counting the number of word-level insertions, deletions, and substitutions required to transform the predicted transcription (hypothesis) into the actual target transcription (reference). The WER formula is expressed as:

$$WER = (S + D + I)/N \quad (6)$$

Here's a more detailed breakdown of the components:

- S is the number of word substitutions (the words in the reference that were incorrectly replaced by words in the hypothesis).
- D is the number of word deletions (the words in reference that were missing in the hypothesis).
- I is the number of word insertions (the words in the hypothesis that were not present in the reference).
- N is the total number of words in the reference transcription.

3.5 Conclusion

In summary, the Tshivenda Speech Corpus is a cornerstone of our study, allowing us to explore speech recognition, linguistic analysis, and model development. This section has laid the foundation for our research, detailing the composition, characteristics, and methodologies employed in analyzing this vital linguistic resource. The chapter concludes with an overview of the evaluation metrics used, specifically the CTC loss and WER. It highlights the importance of these metrics in assessing the model's ability to transcribe Tshivenda speech accurately.

Chapter 4: Experimental Results

4.1 Introduction

In this chapter, I explored the optimal experiment conducted to illustrate scientific concepts effectively. The focus will be on analyzing the results generated from this experiment to understand its outcomes in detail. Additionally, I compared the results of this experiment with those from other similar experiments to highlight the differences and the unique insights gained.

We will analyze the results of the experiment by examining the performance metrics over the course of up to 30 epochs because we couldn’t go beyond those due computational capabilities. Specifically, we will focus on the Word Error Rate (WER), loss, and validation loss to understand how the model improved during training.

4.2 Optimised Hyperparameter Model

Overview of Performance Metrics

We begin by exploring the hyperparameters that are used for the Optimised Hyperparameter Model. Second, by exploring the performance of the speech recognition model, particularly focusing on the calculated metrics such as CTC Loss and Word Error Rate (WER) of the best experiment. These metrics are crucial for evaluating the model's ability to accurately transcribe the Tshivenda language. The findings discussed in this chapter are critical for understanding the efficacy of the methodologies employed and for guiding future improvements in speech recognition systems for under-resourced languages.

Table 4.1 Optimised Hyperparameters

Hyperparameters	Values
Batch_size	32
Learning rate	1xe-4
Drop Rate	0.5
RNN_layer	5
RNN_units	256

Table 4.1 shows the optimized hyperparameters used in developing the end-to-end ASR model for Tshivenda. These hyperparameters were fine-tuned to enhance model performance and generalization. Specifically, a batch size of 32 was selected as it offered a suitable trade-off between model accuracy and computational efficiency, enabling effective use of available hardware resources without overloading memory.

A learning rate of $1\text{x}10^{-4}$ was chosen, a critical parameter that controls the step size during gradient descent. This low learning rate allowed for more stable and gradual convergence to the model's optimal state, reducing the chance of overshooting the minimum and improving overall learning stability across epochs.

To avoid overfitting, a dropout rate of 0.5 was applied, randomly deactivating 50% of the neurons during training. This regularization technique encourages the model to develop more generalizable features by preventing over-reliance on any neuron or layer.

The model architecture included five RNN layers, chosen to ensure sufficient model depth to capture complex temporal dependencies in sequential data. Each RNN layer contains 256 units, providing the model with ample capacity to process the nuances in speech data, which is essential for accurately capturing phonetic information specific to the Tshivenda language.

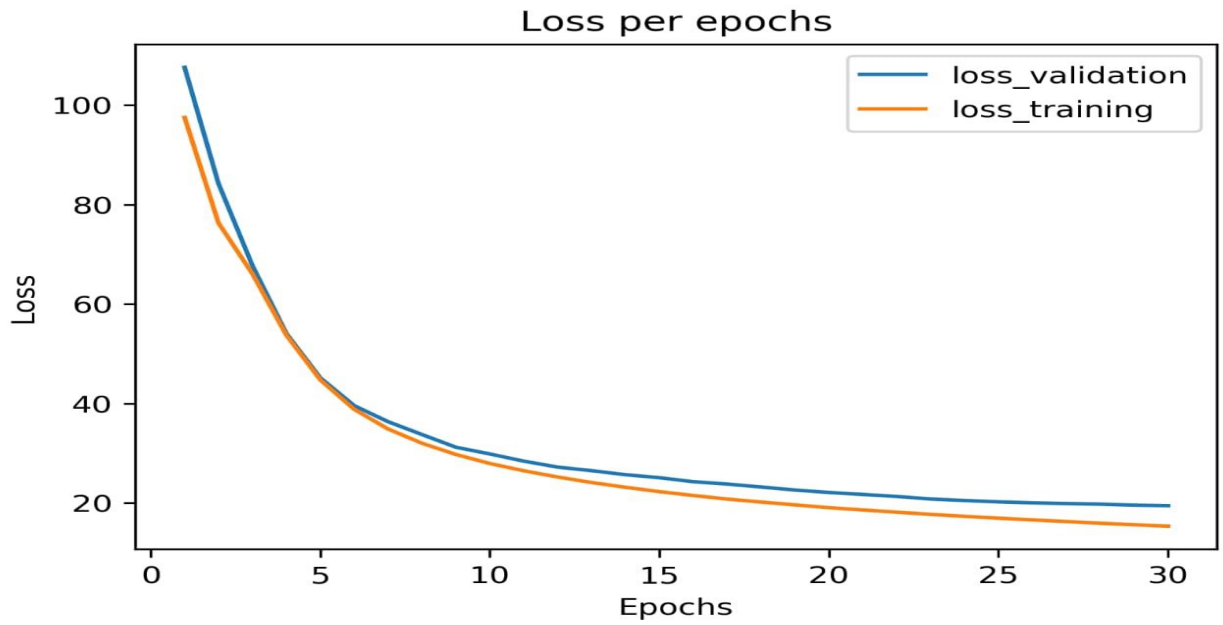

Figure 4.1 Training and Validation Loss per Epoch

Figure 4.1 shows the training and validation loss of the Optimised Hyperparameter Model because the loss metric is crucial as it measures how well the model learns the training data. It is calculated during training and reflects the difference between the model's predictions and the actual target values. A lower loss indicates that the model is making fewer errors on the training data, suggesting effective learning.

At the outset, the model struggled significantly, with a training loss of 97.5 and a validation loss of 107.6 after the first epoch. This high loss suggests that the model had difficulty recognizing and correctly transcribing the speech patterns at this early stage. The predictions during this phase were often incomplete or incorrect, indicating that the model had not yet captured the language features effectively.

As the training progressed, both training and validation losses began to decrease. By epoch 2, the training loss dropped to 76.3, and the validation loss fell to 84.2. This reduction in loss signaled that the model was starting to learn more about the patterns in the Tshivenda speech data. Despite the improvement, many of the predictions were still either incorrect or partially correct, showing that the model had yet to generalize effectively.

By epoch 5, the training loss had further decreased to 53.8, and the validation loss reached 54.1. At this stage, the model demonstrated improved accuracy, with fewer

mistakes in transcriptions. The decreasing losses indicated that the model was becoming better at capturing the complex linguistic features, although some errors in pronunciation or phoneme recognition remained, especially in longer or more complicated sentences.

As the model continued to train, progress became more evident. By epoch 10, the training loss dropped to 29.8, and the validation loss to 31.2. This reduction reflected a significant improvement in the model’s ability to generalize and make accurate transcriptions. While not perfect, the model was now handling shorter and simpler sentences more effectively, though some challenging phrases still led to prediction errors.

In the later epoch, the model's performance continued to improve steadily. By epoch 20, the training loss reached 19.1, with a validation loss of 22.1. At this point, the model was handling a wider range of speech patterns and sentence structures more effectively. Its ability to generalize across different types of sentences had improved, leading to fewer transcription errors, though it occasionally struggled with phoneme differentiation in specific cases.

Finally, by epoch 30, the training loss had decreased to 15.4, and the validation loss to 19.5. At this stage, the model had made substantial progress in learning to transcribe Tshivenda speech. It was now capable of handling complex sentence structures with a higher degree of accuracy, making fewer errors, and producing more reliable transcriptions. The continuing decrease in both losses suggests that the model was successfully learning and generalizing from the data, resulting in improved transcription quality over time.

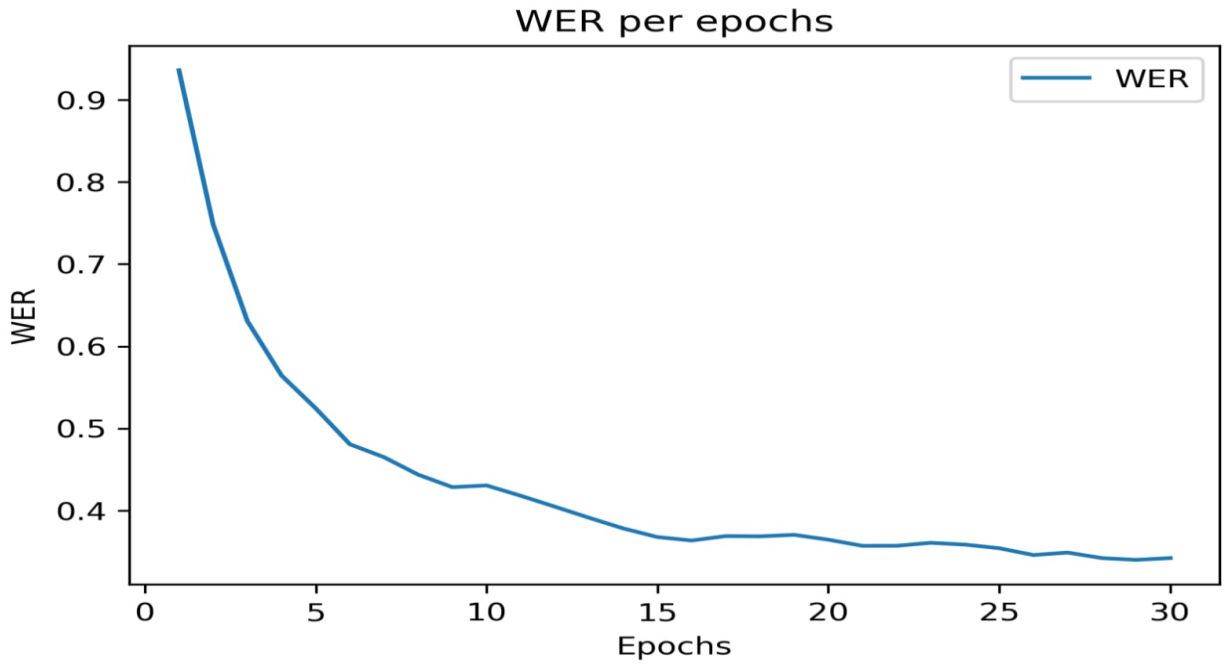

Figure 4.2 The WER per Epoch

Figure 4.2 presents the WER of the Optimized Hyperparameter Model, a vital metric that reflects the model's performance in predicting correct words. WER quantifies the errors (substitutions, deletions, and insertions) made by the model, relative to the total words in the target text. A lower WER signifies fewer errors in the model's predictions, which is crucial for high-accuracy applications such as speech recognition or text translation.

At the beginning of training, epoch 1 exhibited a high WER of 0.9, which is typical for the early stages of model development. The high WER at this stage reflects the model's initial struggle to understand and process the speech data accurately. The high rate of errors suggests that the model had not yet learned the underlying patterns in the dataset, and its predictions were mostly incorrect or far from the intended target. The incorrect predictions at this stage would typically include garbled or meaning outputs, as the model is still learning the fundamental structure of language and has yet to develop meaningful representations of words and phrases. This early phase is often characterized by large errors and inefficient word predictions, as seen in many machine learning models before they undergo significant optimization.

As training advanced, the model's performance improved, and the WER started to decrease. By Epoch 2, the WER had dropped to 0.7, which represents a substantial

improvement. The reduction in WER suggests that the model had begun to learn basic patterns in the data, such as common words, phrases, and speech characteristics. This improvement at the early stages of training is critical because it indicates that the model is rapidly adapting to the features of the speech corpus and learning to minimize the errors in its predictions. The improvement in this phase is often the result of the model learning to better match the spoken words to their corresponding text representations, which reduces errors like substitutions and deletions.

By the time the model reached epoch 3, the WER had further decreased to 0.6, demonstrating that the model was continuing to learn and refine its predictions. This consistent improvement shows that the model was becoming more efficient in recognizing and transcribing spoken words. As the model began to improve, it likely started making fewer deletions and substitutions, particularly with more common and recognizable words. The decreasing WER at this stage highlights the model's ability to generalize better to the training data, which leads to improved accuracy in transcription tasks. This phase also marks the transition from learning basic word patterns to beginning to understand the broader structure of sentences and speech.

As training progressed further, the model made significant strides. By epoch 5, the WER had dropped to 0.5, nearly halving the initial error rate. This represents a critical phase in the training process where the model begins to refine its predictions more systematically, making more accurate distinctions between similar-sounding words and reducing errors caused by noisy input. The model's improvement here would likely be reflected in better predictions of less frequent words or phrases, indicating that it was starting to generalize its knowledge more effectively across different samples. The rapid decline in WER at this stage is a promising sign of the model's capacity to adapt and improve with continued training.

During the middle of the training process, improvements in WER continued, though the rate of decline began to slow as the model approached the limits of its learning capabilities on the given dataset. By Epoch 10, the WER had decreased to 0.4, and by Epoch 15, it further dropped to 0.3. While the improvement was steady, the smaller reductions in WER indicate that the model had already learned many of the patterns present in the data, and

further progress was becoming more incremental. In this phase, the model was likely refining its ability to predict the correct words even in more challenging or ambiguous cases, such as dealing with homophones, variations in pronunciation, or background noise. The model's performance gains during these middle epochs often involve improving its ability to recognize contextual cues, which helps in making more accurate predictions, especially for words that are influenced by their surrounding context.

In the later epochs, particularly from Epoch 20 onward, the WER continued to decline, but the rate of improvement slowed even further. By epoch 30, the WER had settled at 0.3, which represents a significant reduction from the initial value. However, during these later stages of training, the model also experienced minor fluctuations in WER, especially around Epoch 25. These fluctuations could be attributed to the model attempting to generalize better across different samples within the dataset, possibly struggling with variations in speaker accents, background noise, or less frequent words. While the overall trend was still one of improvement, the diminishing returns in the WER reduction suggested that the model was nearing the optimal performance it could achieve with the current dataset and training configuration.

In conclusion, the WER analysis offers a clear perspective on the model’s learning journey throughout training. The initially high WER, followed by a substantial decrease in the early epochs, demonstrates the model’s ability to learn from the training data. The ongoing reduction in WER, with smaller variations in later epochs, indicates the model’s improvement in minimizing errors and its enhanced generalization ability across different data samples. This analysis shows the successful progress of the model, which, despite some fluctuations toward the end, shows significant accuracy improvements over the training period. The final WER value of 0.3 signifies a strong training process and demonstrates the model’s capability to reduce prediction errors, making it well-suited for practical applications like automatic speech recognition in under-resourced languages such as Tshivenda.

Table 4.2 Selected Optimised Hyperparameter Model's Target and Prediction transcriptions

Word Error Rate on NCHLT Testing Data: 0.3934	
Target	ya vhuwalisi yo umetshedzwaho na
Prediction	ya vhulwalisi yo umetshedzwaho na
Target	khungedzelo ya mbetshelo na u
Prediction	khungedzelo ya mbetshelo na u
Target	wa pholisa a dzhioho tshitatamennde
Prediction	wa phoisa a dzhioho tshitatamennde
Target	u oisisa u pfukwa ha
Prediction	u oisisa u pfukwa ha

Table 4.2 shows that the Optimised Hyperparameter WER on the NCHLT Testing data was measured at 0.3934, meaning that approximately 39.34% of the words in the predictions were incorrectly transcribed. This WER score provides a clear indication of the model's performance in recognizing and predicting Tshivenda speech. Given the complexity of the language and the fact that it is under-resourced, this error rate suggests that the model has room for improvement but has also made notable progress. Below is a detailed analysis of some predictions, highlighting the nature of errors and the model's strengths.

In the first example, the target sentence was **"ya vhuwalisi yo umetshedzwaho na"** which translates to **'the evident truth is supported by'**, while the model predicted **"ya vhulwalisi yo umetshedzwaho na"**. Here, the substitution of "vhuwalisi" with "vhulwalisi" is a minor yet significant error. This type of error, where a single character change alters the meaning of the word, illustrates the challenge of achieving high accuracy in word recognition, as even small deviations can lead to incorrect interpretations.

In contrast, the second example, **"khungedzelo ya mbetshelo na u"** which translates to **'the improvement of livelihood and'**, was predicted perfectly by the model without any errors. This accurate transcription showcases the model's ability to recognize certain sentences correctly, demonstrating its potential for improving speech-to-text conversion when it correctly maps input to output without any insertions, deletions, or substitutions.

A more complex error is observed in the third example, where the target sentence was **"wa pholisa a dzhiaho tshitatamennde"** which translate to **‘the healer who takes the lead’**, but the model predicted **"wa phoisa a dziaho tshitandamennde"**. In this case, two errors occurred: "pholisa" was simplified to "phoisa," and "tshitatamennde" became "tshitandamennde." These substitution errors involve multiple characters and alter both the phonetics and, potentially, the meaning of the sentence. This indicates that the model struggles with certain phonetic combinations, which can result in compounded errors across a sentence.

On the other hand, the model successfully predicted the sentence **"u oisisa u pfukwa ha"** which translates to **‘causing disturbance or unrest’** without any errors, demonstrating its ability to handle certain simpler sentences accurately. The absence of errors here suggests that the model is capable of high accuracy when the input is less ambiguous or when it has learned to recognize certain common patterns.

Overall, while the model shows promise in its ability to predict certain sentences correctly, the validation WER of 0.3 and testing WER of 0.4 indicates that there is still a significant number of errors, particularly in the form of substitutions. These errors can have varying impacts, from minor shifts in phonetics to changes that may alter the meaning of a sentence. The detailed analysis of individual cases highlights the areas where the model performs well and where further improvements are needed, particularly in distinguishing between closely related phonemes and in handling more complex sentences.

4.3 All Experiments

To further refine the model and enhance its performance, multiple experiments were conducted, each with a specific configuration of hyperparameters. The goal was to determine the optimal settings that minimize the WER, thereby improving the model's accuracy in predicting the correct words.

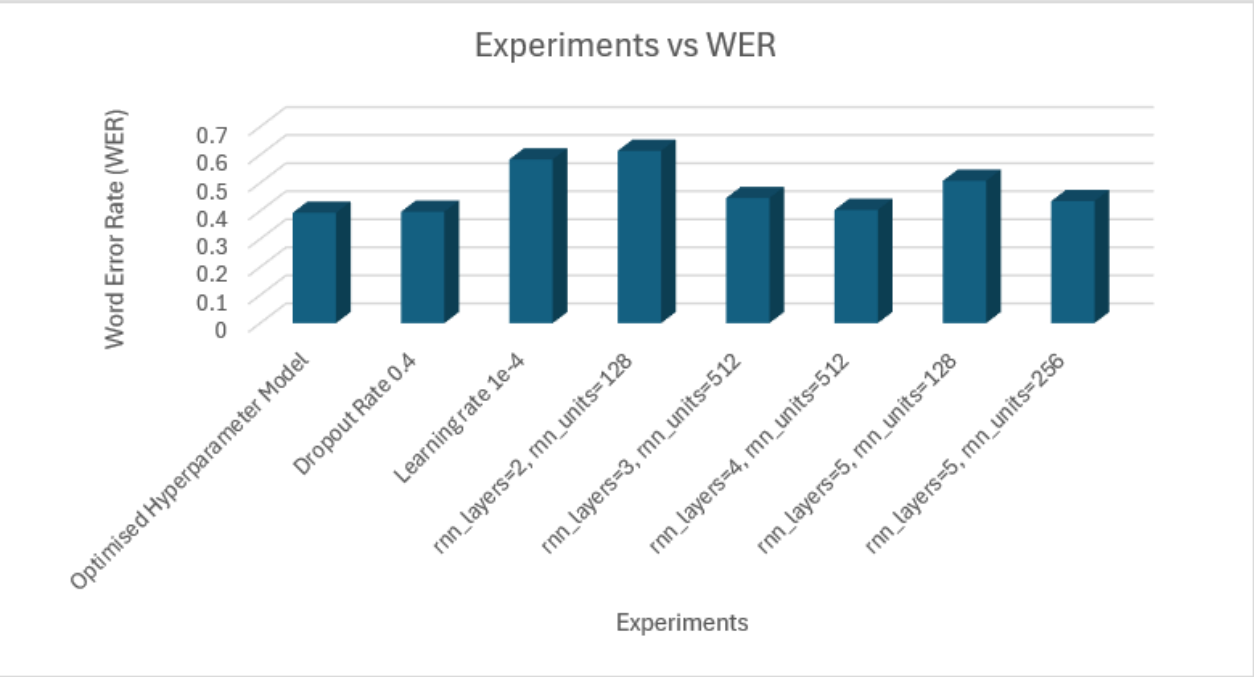

Figure 4.3 Optimised Hyperparameter Model's Target and Prediction Results

We continued to explore the model's performance by analyzing the results of several experiments conducted to optimize the WER. Figure 4.3 shows that these experiments vary in parameters such as the dropout rate, learning rate, and the number of recurrent layers and units in the RNN, offering valuable insights into how these hyperparameters influence the model's accuracy.

The 'Optimised Hyperparameter Model' was compiled by carefully selecting the best-performing hyperparameters from the individual experiments. By analyzing the impact of various configurations, such as the number of layers, units, dropout rate, and learning rate, the most effective values were chosen to form the optimal model. This process of fine-tuning and selecting the best hyperparameter values resulted in the 'Optimised Hyperparameter Model', which showcased the lowest WER, demonstrating the advantage of combining the strengths of each parameter for improved ASR performance.

The experimental results provide insights into the impact of different model configurations and hyperparameters on the WER for the ASR system tailored for Tshivenda, an under-resourced language. The 'Optimised Hyperparameter Model' achieved the best performance with a WER of 0.4, showcasing the model's effectiveness in recognizing and transcribing Tshivenda speech. This suggests that this combination of model architecture

and training settings was the most successful in minimizing errors, making it the most reliable configuration.

In contrast, when a ‘Dropout Rate’ of 0.4 was applied, the WER increased slightly to 0.4. Dropout is often used as a regularization technique to prevent overfitting, and while the performance did not degrade significantly, it suggests that further adjustments to the drop rate could help fine-tune the model’s generalization ability without sacrificing accuracy.

A notable decline in performance was observed when the ‘Learning Rate’ was set to 1e-4, resulting in a significantly higher WER of 0.6. This learning rate likely caused the model to converge too slowly, preventing it from learning effective speech patterns from the data. This emphasizes the critical role that learning rate plays in training deep learning models, especially in speech recognition tasks where the system must learn to discern nuanced auditory patterns.

When examining the effect of the RNN architecture, the results show that varying the number of layers and units had a substantial impact on performance. For instance, when the model was configured with 2 layers and 128 units, the WER was 0.6, which was the highest among all the experiments. This indicates that a shallow RNN with fewer units was insufficient for capturing the complexity of the Tshivenda language, leading to a higher error rate.

However, when the model depth was increased to 3 layers with 512 units, the WER dropped significantly to 0.4, reflecting an improvement in the model's ability to handle more complex features in the speech data. Further increasing the number of layers to 4, while keeping 512 units, resulted in an even lower WER of 0.4036. This shows that deeper architectures with sufficient units can effectively model the complexities of speech, enabling the system to produce more accurate transcriptions.

Interestingly, when the number of layers increased to 5, but with only 128 units, the WER increased again to 0.5. This suggests that while increasing the number of layers can enhance performance, there is a trade-off when the number of units is reduced. A model with too few units per layer might struggle to capture the detailed features of the audio,

even if it is deeper. In this case, having fewer units might have constrained the model's capacity, leading to a higher error rate.

Finally, using 5 layers with 256 units resulted in a WER of 0.4, which is an improvement over the 5-layer model with 128 units but still not as optimal as the 4-layer, 512-unit configuration. This finding highlights the balance needed between depth and the number of units in RNN-based architecture. While deeper networks can help in learning more abstract features, having the right number of units per layer is crucial for maintaining the model’s capacity to process information effectively.

Overall, the experiments demonstrate that the choice of hyperparameters, including the dropout rate, learning rate, number of layers, and the number of units in each layer, plays a critical role in optimizing ASR performance. The best-performing configuration with 4 layers and 512 units shows that a balanced RNN depth with enough units per layer can significantly reduce errors, making it the ideal setup for Tshivenda speech recognition tasks.

4.4 Conclusion

The separate analysis of loss and WER metrics provides comprehensive insights into the model's performance over the 30 training epochs. The steady decline in loss indicates effective learning and optimization of the model on the training data, while the consistent reduction in WER highlights improvements in predictive accuracy. Together, these metrics demonstrate the model's successful training, reflecting its ability to learn from the data and make accurate predictions. The observed fluctuations in later epochs suggest the need for careful monitoring and potential adjustments to avoid overfitting and ensure robust generalization to unseen data.

Chapter 5: Conclusion

5.1 Introduction

In this chapter, we summarize the key findings from the development and evaluation of an ASR system for the Tshivenda language, an under-resourced South African language. Drawing upon the experiments and analyses conducted in Chapter 4, this chapter reflects on the outcomes of the optimized hyperparameter model and discusses the broader implications of the study.

We explored the performance of the ASR system across multiple epochs, focusing on critical evaluation metrics such as WER, and loss. By optimizing various hyperparameters—such as learning rate, dropout rate, and the architecture of recurrent layers—the best-performing model was identified. The steady decrease in both WER and loss metrics over the training period demonstrates the effectiveness of the model in learning and adapting to the linguistic nuances of the Tshivenda language. However, the results also highlight several challenges in accurately transcribing under-resourced languages, particularly with complex phonemes and sentence structures.

The purpose of this conclusion is to provide a synthesis of the findings, discuss the limitations of the research, and propose directions for future work. Ultimately, the development of ASR systems for under-resourced languages like Tshivenda has great potential to enhance language preservation and accessibility, although further refinements are necessary to improve transcription accuracy.

5.2 Summary of Results

The model's training process across 30 epochs showed a consistent improvement in both the loss metrics and WER. Initially, the model struggled with high training and validation losses (97.5 and 107.6, respectively) during the first epoch, coupled with an initial WER of 0.9362. This indicated significant errors in the model's ability to correctly transcribe the Tshivenda speech corpus.

As the training progressed, both losses and WER consistently decreased, reflecting the model's improving ability to capture the complex linguistic features of Tshivenda. By epoch

30, the training loss had dropped to 15.3, and the validation loss was reduced to 19.5. Similarly, the WER declined steadily to 0.3, representing a substantial improvement in transcription accuracy over time.

The optimized hyperparameter model was the best-performing configuration, achieving a final WER of 0.3934 on the NCHLT testing data. This performance demonstrates the model's capability to handle various speech patterns, though certain challenges, such as phoneme recognition and complex sentence structures, still cause errors, particularly in the form of substitutions and deletions.

Additionally, the experiments exploring different hyperparameter settings, such as varying dropout rates and learning rates, further refined the model. The configuration with four layers and 512 recurrent units yielded the lowest WER, highlighting the importance of balanced architectural choices for speech recognition tasks.

In summary, while the ASR model exhibited significant improvements in accuracy, particularly for a low-resource language like Tshivenda, there remains room for further enhancement. The consistent reduction in loss and WER metrics underscores the potential of using deep learning models for under-resourced languages, though some complexities, such as phonetic variations and intricate sentence structures, require additional focus.

5.3 Future Work

While this research has successfully developed an ASR system for Tshivenda using the CTC approach, there are several avenues for future work to enhance both the performance and broader applicability of this system.

5.3.1 Expanding the Dataset

One of the primary limitations of this study was the relatively small size of the NCHLT speech corpus for Tshivenda. Future research could focus on expanding the dataset by collecting more annotated speech samples, including diverse dialects and regional accents of Tshivenda speakers. A larger, more diverse dataset would allow the model to

generalize better and improve its accuracy across different speaking styles and environments.

5.3.2 Incorporating Language Models

Although the CTC approach used in this research produced promising results, integrating a more sophisticated language model could significantly improve transcription accuracy. Future work could explore the use of transformer-based language models, such as BERT or GPT, which can help improve word prediction, especially in scenarios where the ASR model encounters complex or ambiguous sentence structures. The integration of a language model could reduce errors such as substitutions and deletions that were prominent in the current system.

5.3.3 Handling Noisy and Spontaneous Speech

The current model was trained in clean speech data, which may not reflect real-world conditions. Future work should focus on improving the model's robustness by training it on noisy speech datasets, where background noise or overlapping speech is present. Furthermore, handling spontaneous speech with its irregularities, hesitations, and colloquialisms could improve the ASR system’s usability in everyday conversations and broader practical applications.

5.3.4 Exploring Multimodal ASR Systems

Future research could also explore the development of multimodal ASR systems that combine audio with visual data (such as lip movement) to enhance the speech recognition process. By incorporating visual cues, the system could improve accuracy in challenging environments, such as those with significant background noise or when recognizing speakers with speech impairments.

5.4 Limitations

While this research has made significant contributions to the development of an ASR system for Tshivenda, several limitations impacted the performance and scalability of the system. These limitations underscore the challenges faced and suggest areas for further improvement.

5.4.1 System Constraints: Limitation on Training Epochs

Another significant limitation arose due to system resource constraints, which prevented the model from training beyond 30 epochs. While 30 epochs allowed for the model to converge to a reasonable level of accuracy, further improvements in performance could likely have been achieved with additional training. However, due to hardware constraints, increasing the number of training epochs was not feasible. This limitation may have restricted the model's ability to reach its optimal state, potentially resulting in a higher error rate in transcription. More powerful hardware or distributed training methods could help overcome this bottleneck in future iterations.

5.4.2 Limited Dataset Size and Diversity

One of the primary limitations encountered was the restricted size of the NCHLT speech corpus for Tshivenda. While the available dataset provided a foundational training set, its size and diversity were insufficient to fully capture the range of linguistic variations in the Tshivenda-speaking community. The limited number of speakers, accents, and dialectal variations hindered the model's ability to generalize effectively across different Tshivenda-speaking regions, making it less robust in real-world applications where dialectal variations are common.

5.4.3 Sensitivity to Noise and Speaker Variability

The ASR system was primarily trained on clean speech data from the NCHLT corpus, meaning that its robustness in handling real-world noise, varied accents, or non-standard pronunciations remain untested. This limitation reduces the system's reliability when exposed to noisy environments, speech impediments, or distinct regional accents, which are common in everyday usage.

Bibliography

Ali, A. and Renals, S., 2018, July. Word error rate estimation for speech recognition: e-WER. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics,* pp. 20-24.

Audhkhasi, K., Rosenberg, A., Sethy, A., Ramabhadran, B. and Kingsbury, B., 2017. End-to-end ASR-free keyword search from speech. *IEEE Journal of Selected Topics in Signal Processing*, *11*(8), pp.1351-1359.

Barnard, E., Davel, M.H., van Heerden, C., De Wet, F. and Badenhorst, J., 2014. The NCHLT speech corpus of the South African languages. Workshop Spoken Language Technologies for Under-resourced Languages (SLTU). Incomplete specification!
Yu, D. and Deng, L. *Automatic speech recognition* (Vol. 1). Berlin: Springer.

Chang, H.J., Lee, H.Y. and Lee, L.S., 2021. Towards lifelong learning of end-to-end ASR. *arXiv preprint arXiv:2104.01616*.

Chang, J. and Sha, J., 2016. An efficient implementation of 2D convolution in CNN. *IEICE Electronics Express*, pp.13-20161134.

Graves, A., Fernández, S., Gomez, F. and Schmidhuber, J., 2006. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd international conference on Machine learning* pp. 369-376.

Hayashi, T., Watanabe, S., Zhang, Y., Toda, T., Hori, T., Astudillo, R. and Takeda, K., 2018. Back-translation-style data augmentation for end-to-end ASR. In *2018 IEEE Spoken Language Technology Workshop (SLT)* pp. 426-433. IEEE.

Higuchi, Y., Watanabe, S., Chen, N., Ogawa, T. and Kobayashi, T., 2020. Mask CTC: Non-autoregressive end-to-end ASR with CTC and mask predict. *arXiv preprint arXiv:2005.08700*.

Juang, B.H. and Rabiner, L.R., 2005. Automatic speech recognition–a brief history of the technology development. *Georgia Institute of Technology. Atlanta Rutgers University and the University of California. Santa Barbara*.

Koundinya, S., Sharma, H., Sharma, M., Upadhyay, A., Manekar, R., Mukhopadhyay, R., Karmakar, A. and Chaudhury, S., 2018. 2d-3d CNN based architectures for spectral reconstruction from RGB images. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops,* pp. 844-851.

Miller, D.R., Leek, T. and Schwartz, R.M., 1999, August. A hidden Markov model information retrieval system. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval* pp. 214-221.

Mulaudzi, P.A., 1996. *A descriptive analysis of the morphology of the Tshiguvhu dialect of Venda* (Doctoral dissertation, University of South Africa).

Rabiner, L. R. 1989. A tutorial on hidden markov models and selected applications in speech recognition. In *Proceedings of the IEEE*, pp. 257–286.

Salazar, J., Kirchhoff, K. and Huang, Z., 2019. Self-attention networks for connectionist temporal classification in speech recognition. In *ICASSP 2019-2019 IEEE international conference on acoustics, speech and signal processing (ICASSP)* pp. 7115-7119.

Scheidl, H., Fiel, S. and Sablatnig, R., 2018, August. Word beam search: A connectionist temporal classification decoding algorithm. In *2018 16th International Conference on Frontiers in Handwriting Recognition (ICFHR)* pp. 253-258.

Schuster, M. and Paliwal, K.K., Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing*, *45*(11), pp. 2673-2681.

Salazar, J., Kirchhoff, K. and Huang, Z., 2019. Self-attention networks for connectionist temporal classification in speech recognition. In *ICASSP 2019-2019 IEEE international conference on acoustics, speech and signal processing (ICASSP)* (pp. 7115-7119). IEEE.
Soullard, Y., Ruffino, C. and Paquet, T., 2019. Ctcmodel: a keras model for connectionist temporal classification. *arXiv preprint arXiv:1901.07957*.

Scheidl, H., Fiel, S. and Sablatnig, R., 2018, August. Word beam search: A connectionist temporal classification decoding algorithm. In *2018 16th International Conference on Frontiers in Handwriting Recognition (ICFHR)* (pp. 253-258). IEEE.

Wigington, C., Price, B. and Cohen, S., 2019, September. Multi-label connectionist temporal classification. In *2019 International Conference on Document Analysis and Recognition (ICDAR)* (pp. 979-986). IEEE.

Rabiner, L. R. (1989). A tutorial on hidden markov models and selected applications in speech recognition. Proc. IEEE (pp. 257–286). IEEE.
Chen, X., Wei, L. and Xu, J., 2017. House price prediction using LSTM. *arXiv preprint arXiv:1709.08432*.
Li, K., Li, J., Ye, G., Zhao, R. and Gong, Y., 2019, May. Towards code-switching ASR for end-to-end CTC models. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (pp. 6076-6080). IEEE.
Higuchi, Y., Watanabe, S., Chen, N., Ogawa, T. and Kobayashi, T., 2020. Mask CTC: Non-autoregressive end-to-end ASR with CTC and mask predict. *arXiv preprint arXiv:2005.08700*.
Watanabe, S., Hori, T., Kim, S., Hershey, J.R. and Hayashi, T., 2017. Hybrid CTC/attention architecture for end-to-end speech recognition. *IEEE Journal of Selected Topics in Signal Processing*, *11*(8), (pp.1240-1253).
Song, W. and Cai, J., 2015. End-to-end deep neural network for automatic speech recognition. *Standford CS224D Reports*.
Padmanabhan, J. and Johnson Premkumar, M.J., 2015. Machine learning in automatic speech recognition: A survey. *IETE Technical Review*, *32*(4), (pp. 240-251).
Zhang, W., Cui, X., Finkler, U., Kingsbury, B., Saon, G., Kung, D. and Picheny, M., 2019, May. Distributed deep learning strategies for automatic speech recognition. In *ICASSP*

2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) (pp. 5706-5710). IEEE.

Malik, M., Malik, M.K., Mehmood, K. and Makhdoom, I., 2021. Automatic Speech Recognition: a survey. *Multimedia Tools and Applications*, *80*(6), pp. 9411-9457.

Vadwala AY, Suthar KA, Karmakar YA, Pandya N (2017) Survey paper on different speech recognition algorithm: challenges and techniques. Int J Comput Appl 175(1), pp. 31–36

Benzeghiba, M., De Mori, R., Deroo, O., Dupont, S., Erbes, T., Jouvet, D., Fissore, L., Laface, P., Mertins, A., Ris, C. and Rose, R., 2007. Automatic speech recognition and speech variability: A review. *Speech communication*, *49*(10-11), pp. 763-786.

Montavon, G., Samek, W. and Müller, K.R., 2018. Methods for interpreting and understanding deep neural networks. *Digital Signal Processing*, *73*, (pp. 1-15).

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

Yosinski, J., Clune, J., Bengio, Y. and Lipson, H., 2014. How transferable are features in deep neural networks. *Advances in neural information processing systems*, pp *27*.

Szegedy, C., Toshev, A. and Erhan, D., 2013. Deep neural networks for object detection. *Advances in neural information processing systems*, *26*.

Krogh, A., 2008. What are Artificial Neural Networks. *Nature biotechnology*, *26*(2), pp.195-197.

Abraham, A., 2005. Artificial Neural Networks. *Handbook of measuring system design*.

Yang, G.R. and Wang, X.J., 2020. Artificial neural networks for neuroscientists: A primer. *Neuron*, *107*(6), pp.1048-1070.

Thamarai, M. and Malarvizhi, S.P., 2020. House Price Prediction Modeling Using Machine Learning. *International Journal of Information Engineering & Electronic Business*

Avanijaa, J., 2021. Prediction of House Price Using XGBoost Regression Algorithm. *Turkish Journal of Computer and Mathematics Education (TURCOMAT)*, *12*(2), pp. 2151-2155.

Alfiyatin, A.N., Febrita, R.E., Taufiq, H. and Mahmudy, W.F., 2017. Modeling house price prediction using regression analysis and particle swarm optimization. *International Journal of Advanced Computer Science and Applications*

Limsombunchai, V., 2004, June. House price prediction: hedonic price model vs. artificial neural network. In *New Zealand agricultural and resource economics society conference*

Thad Starner, Alex Pentland. Real-Time American Sign Language Visual Recognition From Video Using Hidden Markov Models. Master's Thesis, MIT, Feb 1995, Program in Media Arts

Rabiner, L. and Juang, B., 1986. An introduction to hidden Markov models. *IEEE ASSP magazine*, *3*(1), pp.4-16.

Bahl, L., Brown, P., De Souza, P. and Mercer, R., 1986, April. Maximum mutual information estimation of hidden Markov model parameters for speech recognition. In *ICASSP'86. IEEE International Conference On Acoustics, Speech, And Signal Processing* (Vol. 11, pp. 49-52). IEEE.

Rabiner, L.R., 1989. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, *77*(2), pp.257-286.

Sherstinsky, A., 2020. Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network. *Physica D: Nonlinear Phenomena*, *404*, p.132306.

Shivakumar, P.G., Potamianos, A., Lee, S. and Narayanan, S.S., 2014, September. Improving speech recognition for children using acoustic adaptation and pronunciation modeling. In Workshop on Child Computer Interaction (WOCCI) (pp. 15-19).

Cremelie, N. and Martens, J.P., 1999. In search of better pronunciation models for speech recognition. Speech Communication, 29(2-4), pp.115-136.

Kanokphara, S., Tesprasit, V. and Thongprasirt, R., 2003, April. Pronunciation variation speech recognition without dictionary modification on sparse database. In 2003 IEEE International Conference on Acoustics, Speech, and Signal Processing, 2003. Proceedings.(ICASSP'03). (Vol. 1, pp. 1). IEEE.

Plauche, M., Nallasamy, U., Pal, J., Wooters, C. and Ramachandran, D., 2006, May. Speech recognition for illiterate access to information and technology. In 2006 International conference on information and communication technologies and development (pp. 83-92). IEEE.

Chan, W., Jaitly, N., Le, Q. and Vinyals, O., 2016, March. Listen, attend and spell: A neural network for large vocabulary conversational speech recognition. In 2016 IEEE International Conference On Acoustics, Speech And Signal Processing (ICASSP) (pp. 4960-4964). IEEE.

Ortmanns, S., Ney, H. and Eiden, A., 1996, October. Language-model look-ahead for large vocabulary speech recognition. In Proceeding of Fourth International Conference on Spoken Language Processing. ICSLP'96 (Vol. 4, pp. 2095-2098). IEEE.

Jelinek, F., Merialdo, B., Roukos, S. and Strauss, M., 1991. A dynamic language model for speech recognition. In Speech and Natural Language: Proceedings of a Workshop Held at Pacific Grove, California, February 19-22, 1991.

Ravishankar, M.K., 1996. Efficient Algorithms for Speech Recognition. Carnegie-Mellon Univ Pittsburgh pa Dept of Computer Science.

Watanabe, S., Hori, T., Kim, S., Hershey, J.R. and Hayashi, T., 2017. Hybrid CTC/attention architecture for end-to-end speech recognition. IEEE Journal of Selected Topics in Signal Processing, 11(8), pp.1240-1253.

Miao, H., Cheng, G., Zhang, P., Li, T. and Yan, Y., 2019, September. Online Hybrid CTC/Attention Architecture for End-to-End Speech Recognition. In Interspeech (pp. 2623-2627).

Cui, J., Weng, C., Wang, G., Wang, J., Wang, P., Yu, C., Su, D. and Yu, D., 2018, December. Improving attention-based end-to-end ASR systems with sequence-based loss functions. In 2018 IEEE Spoken Language Technology Workshop (SLT) (pp. 353-360). IEEE.

Cremelie, N., 2014. *Title of the Work*. *Publication/Conference Name*, Volume(Issue).

Cui, J., Weng, C., Wang, G., Wang, J., Wang, P., Yu, C., Su, D., and Yu, D., 2018, December. Improving attention-based end-to-end ASR systems with sequence-based loss functions. In 2018 IEEE Spoken Language Technology Workshop (SLT) (pp. 353-360). IEEE.

Sung, W. T., Kang, H. W., & Hsiao, S. J. (2023). Speech Recognition via CTC-CNN Model. *Computers, materials & continua*, *76*(3).