Applicability of Transfer Learning on End-to-End Sinhala Speech Recognition

W.D.H. Pushpakumara

2024

Applicability of Transfer Learning on End-to-End Sinhala Speech Recognition

W.D.H. Pushpakumara
Index Number: 19001258

Supervisor: Dr. B.H.R. Pushpananda
Co-Supervisor: Mrs. A.L. Nanayakkara

May 2024

Submitted in partial fulfillment of the requirements of the B.Sc. (Honours) in Computer Science Final Year Project

Declaration

I certify that this dissertation does not incorporate, without acknowledgement, any material previously submitted for a degree or diploma in any university and to the best of my knowledge and belief, it does not contain any material previously published or written by another person or myself except where due reference is made in the text. I also hereby give consent for my dissertation, if accepted, be made available for photocopying and for interlibrary loans, and for the title and abstract to be made available to outside organizations.

Candidate Name: W.D.H. Pushpakumara

[illegible]

Signature of the Candidate Date: 29-09-2024

This is to certify that this dissertation is based on the work of Mr. W.D.H. Pushpakumara under my supervision. The thesis has been prepared according to the format stipulated and is of acceptable standard.

Principle Supervisor's Name: Dr. B.H.R. Pushpananda

Randil

Signature of the Principle Supervisor Date: 29-09-2024

Co-Supervisor's Name: Mrs. A.L. Nanayakkara

[illegible]

Signature of the Co-Supervisor Date: 29-09-2024

Abstract

Automatic Speech Recognition (ASR) is a rapidly evolving area within Natural Language Processing (NLP), addressing a range of linguistic challenges. While ASR technologies have made significant strides through various models, including Hidden Markov Models (HMMs), Gaussian Mixture Models (GMMs), and more recently, Deep Neural Networks (DNNs), Convolutional Neural Networks (CNNs), and Recurrent Neural Networks (RNNs), certain languages like Sinhala face specific limitations. One major challenge for Sinhala ASR development is the lack of sufficient labeled speech data, which makes it difficult and costly to build accurate models.

This thesis explores a transfer learning-based approach to mitigate the data scarcity problem in Sinhala ASR. Specifically, the study leverages the XLS-R model developed by Babu et al. (2021) as the source model, using its pre-learned speech representations to fine-tune a Sinhala ASR model. Two distinct datasets, differing in their lexical composition, were used to evaluate the model's performance. The proposed model achieved Word Error Rates (WER) of 33.78% and 38.31% on the two datasets, respectively. To further enhance transcription accuracy, post-processing steps, including spell correction and word boundary correction algorithms, were applied, resulting in improved WERs of 24.28% and 36.6%.

While the baseline model performed better on the first dataset, a relative WER reduction of 10.07% was observed on the second dataset. An analysis of the generated transcriptions indicates that the proposed model produces results that are acceptable in practical applications, highlighting its potential to improve ASR performance for under-resourced languages like Sinhala.

Keywords: Sinhala, speech recognition, transfer learning, XLS-R, post-processing

Acknowledgement

I would like to express my deepest gratitude to the University of Colombo School of Computing (UCSC) for granting me the opportunity and essential resources to pursue my research.

I would like to express my sincere gratitude to my supervisor, Dr. Randil Pushpananda, for his invaluable guidance, knowledge, and encouragement during the whole length of this project. His contributions have played a crucial role in molding the final outcome. I am deeply appreciative of Mrs. Lakshika Nanayakkara for her essential perspectives and assistance as my co-supervisor.

I express my heartfelt thanks to Mr. Buddhi Gamage and Ms. Ashmari Pramodya for their guidance and support, which have been extremely beneficial in navigating the complexities of this research.

I am also grateful to my fellow batch mates, who helped me in preparing the speech dataset for testing which was essential to the accomplishment of this research.

Finally, I would like to express my heartfelt gratitude to all individuals who took part in the evaluation procedure and offered excellent suggestions and criticism, greatly contributing to the improvement of this research.

Contents

List of Figures

List of Tables

Acronyms

AI Artificial Intelligence

ASR Automatic Speech Recognition

BERT Bidirectional Encoder Representations from Transformers

CER Character Error Rate

CNN Convolutional Neural Network

CSV Comma Separated Values

CTC Connectionist Temporal Classification

GMM Gaussian Mixture Models

HMM Hidden Markov Model

LM Language Model

LSTM Long Short Term Memory

LTRL Language Technology Research Lab

MPE Minimum Phone Error Rate

ML Machine Learning

UCSC University of Colombo School of Computing

WER Word Error Rate

Chapter 1

Introduction

1.1 Background to the Research

Over the years, Automatic Speech Recognition (ASR) technology has undergone substantial changes in its approach, transitioning from early stages and traditional statistical models to more advanced neural network based models.

The earliest project which can be considered as ASR was the “Audrey” technology invented by the researchers at Bell laboratories in the United States. It could only recognize spoken numerical digits.

Then, after several research programs, in 1980’s the field of ASR faced a major paradigm shift to statistical approaches (Arora and R. P. Singh 2012). Hidden Markov Models (HMMs) were introduced as a result of this paradigm shift (Juang and L. R. Rabiner 1991). For over a 30 years, it has been the major methodology which the ASR systems were based on. Even today, most of the practical ASR systems are based on statistical methods (Arora and R. P. Singh 2012; D. Wang et al. 2019).

Subsequently, in the 2010s, Deep Neural Networks (DNNs) emerged as a result of the progress in deep learning. The introduction of DNN-HMM based ASR models, which utilized DNNs to construct the acoustic models, led to the emergence of the most advanced ASR models. Next, a paradigm change occurred with end-to-end ASR models, which translated acoustic information directly into a word sequence (D. Wang et al. 2019). Since then, up till now, Recurrent Neural Networks (RNNs), Long-Short Term Memory (LSTM) networks, Convolutional Neural Networks (CNNs), and Transformers have been employed to enhance the accuracy and develop more sophisticated ASR systems (Li et al. 2022).

ASR systems for the Sinhala language have likewise undergone a similar path to those of other languages. A continuous speech recognizer was developed in 2011 using an HMM-GMM based approach. This approach resulted in a recognition accuracy of 96.14% (Nadungodage and Weerasinghe 2011). Karunathilaka et al. (2020), have experimented with pre-trained DNN, DNN, Time Delay Neural Networks (TDNN), TDNN+LSTM to enhance the acoustic modeling process. In 2021, Gamage, Pushpananda, Nadungodage, et al. (2021) developed an ASR model employing the end-to-end architecture specifically for the Sinhala language for the first time. Nevertheless, the scarcity and high cost of gathering speech data for the Sinhala language, which is considered a low-resource language, has consistently hindered researchers from developing an optimal ASR system.

As a solution for the problem of having limited resources in terms of data, researchers started using the method called transfer learning (Bozinovski and Fulgosi 1976). Transfer learning is a machine learning method where the knowledge learned from one task is applied to another separate or same but related task (Tan et al. 2018). In the field of ASR, when applying transfer learning, there are two types of source models which can be used to transfer knowledge from. Mono-lingual models (trained with only a single language) and Multi-lingual models (trained with several languages). Both source models have been employed by researchers to examine their performance in terms of accuracy and the resource requirements and have succeeded to gain excellent outcomes (Huang et al. 2013; Yu et al. 2019). There are only a few research done for Sinhala language using transfer learning (Nanayakkara and Weerasinghe 2023; Karunanayake et al. 2019). None of them have experimented with a multi-lingual model as the source model. Therefore it will be advantageous to explore the applicability of employing a multi-lingual model as the source model to create an ASR system for Sinhala with an acceptable accuracy while utilizing the available data.

XLS-R is a large-scale model for cross-lingual speech representation learning which is trained using 128 languages including Sinhala (Babu et al. 2021). It has been effectively adopted by the ASR community recently as a multi-lingual source model for constructing ASR systems for low-resourced languages (Krishna et al. 2021; Arisaputra et al. 2024).

1.2 Problem Statement

The problem statement lies in the under-utilization of effective methodologies, particularly multi-lingual transfer learning (TL), in the domain of Sinhala Automatic Speech Recognition (ASR). Despite the proven efficacy of multi-lingual TL in enhancing ASR performance for low-resource languages, its applicability to Sinhala language remains largely unexplored. This deficiency represents a significant gap in research, hindering the advancement of Sinhala ASR capabilities and limiting its potential to achieve competitive accuracy levels comparable to well-resourced languages. Thus, there is an urgent need to investigate and leverage multi-lingual TL techniques to address the unique challenges of Sinhala ASR, thereby paving the way for improved accuracy and accessibility in Sinhala speech recognition systems.

1.3 Research Aim, Questions and Objectives

This section describes an overview of the proposed research project's research aim, questions, and objectives.

1.3.1 Research Aim

To provide a more accurate and efficient ASR system for Sinhala language to compete with other high-resourced ASR systems and to circumvent the limitation of scarcity in linguistic resources.

1.3.2 Research Questions

RQ 1 How can multi-lingual transfer learning models be used to improve Sinhala ASR?

This research aims to enhance Sinhala ASR performance using multi-lingual transfer learning models addressing resource limitations. By improving the existing standards and reducing the gap in terms of accuracy and robustness with high-resourced languages, it will pave the way for future researchers to further bridge the disparity. Through experimental evaluations and comparisons with baseline models, this research question aims to demonstrate the effectiveness of multi-lingual models in improving the accuracy and performance of Sinhala ASR. The findings of this study can contribute to the development of more robust and accurate ASR systems for low-resource languages like Sinhala.

RQ 2 How to optimize a Sinhala ASR model to achieve the best accuracy while being resource efficient?

After answering the above research question, this research question refers to finding out any limitations which might arise after developing a multi-lingual transfer learning model for Sinhala language. This research question is answered by finding out if there are any techniques to mitigate these limitations, and thereby optimizing the ASR model while being resource efficient.

1.3.3 Research Objectives

RO 1 Investigate the effectiveness in choosing a multi-lingual pre-trained model as the source model vs. choosing a mono-lingual model.

RO 2 Experimenting to find the optimal combination of source languages and optimal amount of training data to improve the accuracy while keeping the requirement of computational power within availability.

RO 3 Developing an accurate and efficient ASR model for Sinhala language by incorporating all the findings from the above objectives and compare it with a baseline model to evaluate the model.

RO 4 Investigate the limitations and other issues that could arise during and after the process of developing the ASR system.

RO 5 Investigate the solutions for the identified limitations.

1.4 Significance of the Project

The proposed project holds great significance in the both the field of computer science and society. From a computer science perspective, it contributes to the field of ASR by tackling the crucial challenge of limited annotated data in low-resource languages like Sinhala. By exploring and implementing multi-lingual learning this research aims to develop more accurate and efficient ASR models for the Sinhala language.
From a societal perspective, the impact of this project is substantial. The availability of accurate and efficient ASR systems for the Sinhala language has profound implications for communication, accessibility, and inclusion in Sri Lanka. It can empower individuals and communities by enabling natural language interactions with technology, facilitating access to information and services, and bridging language barriers. This research, therefore, contributes to the broader goal of promoting linguistic diversity, cultural preservation, and equal opportunities for individuals speaking Sinhala.

1.5 Research Approach And Methodology

1. **Recreation of baseline models to compare**

   The very first approach is to recreate the baseline models that is going to be used as benchmarks for comparison.

   - End-to-end model introduced in (Gamage, Pushpananda, Nadungodage, et al. 2021)
   - Mono-lingual transfer learned model introduced in (Nanayakkara and Weerasinghe 2023) with and without data augmentation.

2. **Implementing the multi-lingual model**

   As the next step, the multi-lingual model will be trained using a suitable pre-trained model(s). Experimentation will be done to end up with an optimal ASR system. With the size of the pre-trained model and the available resources, the optimal methods will be chosen for the model training. For an example, according to Dalmia et al. (2018), full model adaptations provide more accuracy. However, it requires more computational power. Therefore, optimal methodologies will be chosen by comparing each other.

3. **Identifying the limitations and problems with the resultant models**

   Identify any limitation or recurring problems after evaluating the models

4. **Experimenting with various techniques to overcome the identified limitations and derive an optimal model for Sinhala ASR within the technical capabilities**

Analyze the nature of the identified limitations and research to find out any solutions to mitigate them.

1.6 Outline of the Dissertation

The dissertation is organized as follows. In chapter 2, a comprehensive literature review has been conducted to outline the evolution of ASR and the gap between Sinhala ASR and other ASR systems in terms of the technologies used. Chapter 3 contains in-depth information on the technologies, architectures and algorithms used in this research. In chapter 4, the step by step approach taken in this research has been outlined. Chapter 5 displays the results from the experimentation done throughout the research and an analysis of the results. Finally, Chapter 6 provides the conclusions drawn from the research.

1.7 Scope Including Delimitation

In Scope

This study covers the following aspects.

- Use multi-lingual learning to develop an accurate ASR model for Sinhala language.
- Fine tuning the developed model to optimize the performance.
- Finding out the best possible language combination as source languages.
- Comparing with an existing mono-lingual model to evaluate the model.

Out of Scope

This study will **not** cover the following aspect.

- Fine tuning the existing baseline models when comparing.
- Using only pre-trained models rather than training my own models which may be too time consuming.

Chapter 2

Literature Review

2.1 History of ASR

The origins of Automatic Speech Recognition (ASR) can be traced back to the mid-20$^{\text{th}}$ century when initial studies were conducted to comprehend basic spoken phrases and commands. These endeavors were somewhat rudimentary as a result of the constrained technological resources accessible during that period. Nevertheless, as technology advanced, so did ASR. In the late 20$^{\text{th}}$ century, statistical models, particularly Hidden Markov Models (HMMs), gained popularity. Hidden Markov Models (HMMs) proved to be successful in capturing the temporal patterns of speech sounds, resulting in improved accuracy in speech recognition. Nevertheless, it was only in recent years that Automatic Speech Recognition (ASR) had a significant advancement with the use of neural networks. Recurrent neural networks (RNNs) and convolutional neural networks (CNNs) have greatly enhanced automatic speech recognition (ASR) performance. They have even surpassed previous methods and achieved accuracy similar to that of humans in several tests. This development underscores the continuous progress in Automatic Speech Recognition (ASR), propelled by enhancements in both computational capabilities and methods (Arora and R. P. Singh 2012).

2.1.1 Early Stages

Speech recognition has a long and significant history that can be traced back to the early 1920s when Radio Rex, the first voice recognition toy, was invented. During the World Fair in New York, Bell Labs displayed a voice synthesis machine, but eventually abandoned further work after incorrectly believing that success required artificial intelligence (AI). In the 1950s, researchers delved into fundamental phonetic-acoustic ideas to build automatic speech recognition (ASR) systems. Initial systems focused on analyzing spectral resonances, notably in vowel sounds (Arora and R. P. Singh 2012).

At Bell Labs, Davis et al. (1952) created a digit recognition system for a single speaker by predicting formant frequencies during vowel regions. Olson and Belar (1956), at RCA Labs, constructed a 10-syllable recognizer for a single speaker, while J. W. Forgie and C. D. Forgie (1959), at MIT Lincoln Lab, developed a speaker-independent 10-vowel recognizer by monitoring spectral resonances. Fry (1959) and Denes (1959) attempted a phoneme recognizer for four vowels and nine consonants at University College, England, utilizing spectrum analyzers and pattern matchers (Arora and R. P. Singh 2012).

In the 1960s and 1970s, Japanese labs entered this field, building special-purpose hardware due to computer restrictions. Nagata et al. (1964), at Tokyo's Radio Research Lab, created a hardware vowel recognizer, and a hardware phoneme recognizer was built at the Kyoto University by Sakai (1962). Nagata et al. (1964), at NEC Labs, produced a digit recognizer in 1963, commencing a productive research phase. By the 1970s, research switched to isolated word recognition, with IBM working on large vocabulary speech recognition and AT&T Bell Labs experimenting with speaker-independent systems (L. Rabiner et al. 1979). Numerous clustering algorithms were applied to uncover different patterns for speaker-independent recognition, leading to widespread acceptance of these techniques.

2.1.2 Statistical Models

In 1980, research primarily centered on connected word speech recognition. Early in the decade, marked a shift in technology from template-based methods to statistical modeling, particularly the adoption of Hidden Markov Models (HMM) in speech research (Arora and R. P. Singh 2012).

HMM-based models consists of three key models: the acoustic model, pronunciation model, and language model. The acoustic model serves as the bridge between voice input and feature sequences, typically phonemes or sub-phonemes, by statistically analyzing auditory patterns in spoken language, forming the basis for accurate transcription. Complementing this, the pronunciation model establishes the relationship between phonemes and their corresponding graphemes, aiding in mapping spoken words to written forms. Finally, the language model provides probabilistic distributions over word sequences, offering contextual knowledge crucial for precise and contextually relevant transcription by estimating the likelihood of specific words occurring in a given context. Together, these models form the backbone of ASR systems, enabling the accurate conversion of spoken language into text (D. Wang et al. 2019).

Then, GMM-HMM became prominent during this period. GMMs were introduced as a technique to model the acoustic features of speech. GMMs depict the probability distribution of feature vectors collected from speech data. HMMs were utilized to model the temporal transitions between these GMMs. This approach produced promising results and became a standard in ASR systems. One famous system, SPHINX, designed by Kai-Fu Lee at Carnegie Mellon University, uses HMM to describe speech patterns over time and Gaussian Mixture Models (GMM) to reflect the likelihood of observing distinct speech states (Lee 1988). This strategy constituted a significant breakthrough in Large Vocabulary Continuous Speech Recognition (LVCSR), remaining dominant until the development of deep learning approaches. Despite its success, HMM-GMM faced difficulties, especially in reliably transcribing common interactions like phone calls and conferences, motivating the development of other approaches (D. Wang et al. 2019).

2.1.3 Neural Networks

In the mid-2000s, the emergence of deep neural networks (DNNs) revolutionized acoustic modeling in ASR. Particularly convolutional neural networks (CNNs) and recurrent neural networks (RNNs), showed superior performance in acoustic modeling and feature extraction compared to traditional methods. DNNs had various advantages over GMMs, including the ability to automatically learn hierarchical features from raw data and catch complicated patterns in speech signals. In 2011, Dahl et al. (2011) proposed an HMM combined with context-based DNN which named context-dependent(CD)-DNN-HMM. The proposed CD-DNN-HMMs demonstrated superior performance compared to conventional Context-Dependent Gaussian Mixture Model Hidden Markov Models (CD-GMM-HMMs) in large-vocabulary speech recognition tasks. The CD-DNN-HMMs achieved an absolute sentence accuracy improvement of 5.8% and 9.2% (or relative error reduction of 16.0% and 23.2%) over CD-GMM-HMMs trained using minimum phone error rate (MPE) and maximum-likelihood (ML) criteria, respectively. This significant improvement showcases the effectiveness of the proposed model in enhancing speech recognition accuracy.

The HMM-DNN model faces challenges like forced data segmentation alignment, independent hypotheses, and separate module training inherited from HMM. However, deep learning has permitted the construction of end-to-end models that immediately translate input audio signals into transcriptions, streamlining the architecture and training process for greater precision and speed. In order to mitigate the challenges of HMM-DNN based models, end-to-end models were developed. In contrast with HMM-DNN models, the end-to-end model offers simplification, joint training, direct output, and eliminates the need for forced data alignment, among other advantages (D. Wang et al. 2019).

End-to-end speech recognition models typically consist of an encoder, aligner, and decoder. However, these components may not be clearly distinguishable as in traditional modular systems. Unlike HMM-based models with multiple modules, end-to-end models use a single deep network to directly map acoustic signals to label sequences without intermediate states. This eliminates the need for posterior processing. The key characteristics of end-to-end LVCSR compared to HMM-based models include joint training of multiple modules in one network, direct mapping from acoustic input to text output without additional processing, and the absence of internal representations for pronunciation in character-level models (D. Wang et al. 2019).

2.2 Low-resource Speech Recognition

A particular language in which the collection of labeled speech data to build an ASR model is expensive and difficult due to the scarcity of data is usually referred to as "low-resourced" in the ASR community. Sinhala is considered to be such a language. Despite being a low-resource language, Sinhala ASR models has witnessed a similar evolution in terms of technologies throughout the years.

In 2011, Nadungodage and Weerasinghe (2011) developed a continuous speech recognizer for Sinhala using an acoustic model based on an HMM. Although they have managed to achieve an accuracy of 96.14%, the model had a limitation of being speaker dependent.

In 2020, Karunathilaka et al. (2020) experimented with several deep neural architectures such as pre-trained DNN, DNN, Time Delay Neural Network (TDNN), TDNN + Long Short Term Memory (LSTM) to enhance the acoustic modeling process. Their TDNN based model resulted in 7.48% better word error rate (WER) compared to their baseline models.

In 2021, a research was conducted by Gamage, Pushpananda, Nadungodage, et al. (2021) to build an end-to-end ASR model for Sinhala for the first time which provided with a result of 28.55% WER.

Eventually, due to the nature of being a low-resource language, Sinhala ASR systems had to move on to different techniques such as transfer learning to mitigate this issue.

2.3 Transfer Learning

Transfer learning involves leveraging knowledge gained from one or multiple source tasks to aid in the completion of a target task (Pan and Yang 2009). Figure 2.1 shows the illustration of the learning processes of traditional machine learning (ML) and transfer learning.

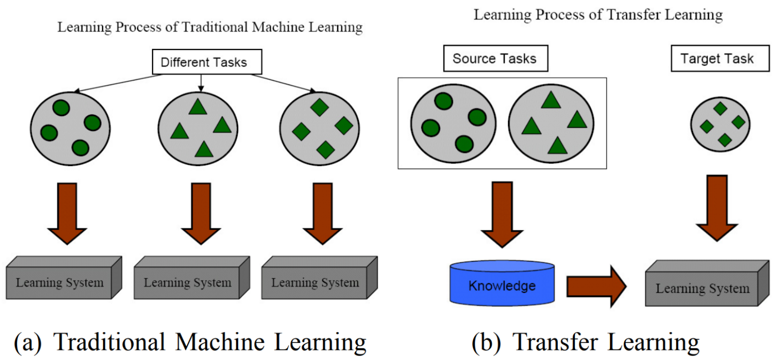


Figure 2.1: Traditional machine learning vs. transfer learning (Pan and Yang 2009)

As discussed earlier, Sinhala being a low-resourced language, has become a limitation for the ASR community to build improved ASR models using traditional machine learning. Transfer learning provides a solution for this problem through allowing to transfer the knowledge from a source task to a target task. Figure 2.2 shows an architecture of a cross-lingual knowledge transfer.

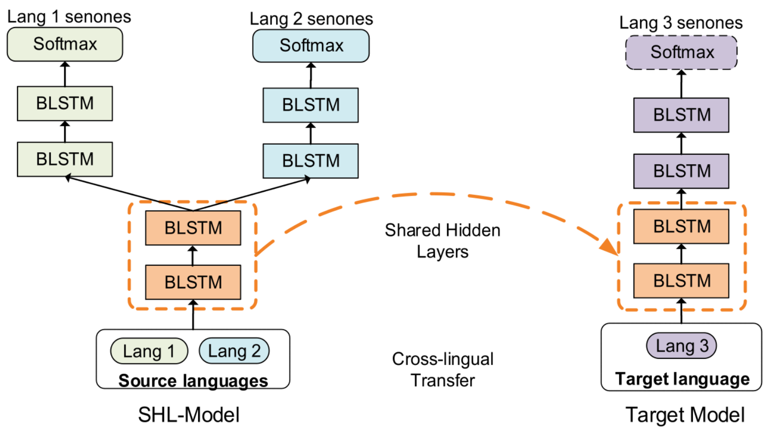


Figure 2.2: An architecture of a cross-lingual knowledge transfer learning model (Yi et al. 2018).

When using transfer learning for speech recognition, there are two methods to implement it based on the fact that the model which the knowledge is transferred from (source model) is trained using a single language or multiple languages. In this research, they are referred to as mono-lingual models and multi-lingual models respectively.

Very few studies have focused on mono-lingual models for the Sinhala language. Karunanayake et al. (2019) developed a mono-lingual ASR model by transferring knowledge from a pre-trained English model to Sinhala and Tamil languages using the DeepSpeech engine. They achieved notable accuracy improvements, with their best model reaching 63.23% accuracy for Sinhala compared to 93.16% for the mono-lingual model.

Nanayakkara and Weerasinghe (2023) conducted another significant research solely on Sinhala, aiming to build an ASR model with the lowest word error rate (WER) possible through mono-lingual transfer learning using DeepSpeech. They incorporated data augmentation to further enhance WER. Their efforts resulted in a WER of 22.92% without data augmentation, surpassing the accuracy of the best available end-to-end models. With data augmentation, they achieved a remarkable 17.18% WER, indicating significant progress in the field.

Only mono-lingual source models have been experimented on Sinhala ASR as of now. The applicability of transferring knowledge from a multi-lingual source model for Sinhala ASR remains unexplored. However, many other research have been done on other low-resource languages using multi-lingual source models.

The authors of Dalmia et al. (2018) have implemented techniques for multi-lingual and cross-lingual speech recognition, aiming to aid low-resource scenarios, bootstrap systems, and facilitate analysis of new languages and domains. They particularly focus on end-to-end approaches, especially sequence-based techniques, due to their simplicity and effectiveness. They demonstrate that end-to-end multi-lingual training of sequence models using Connectionist Temporal Classification (CTC) loss is effective, even without traditional multi-lingual bottleneck feature extractors as front-ends. Their model shows significant improvement in performance on Babel languages, achieving over a 6% absolute reduction in word/-phoneme error rate compared to mono-lingual systems built under the same conditions for those languages. Additionally, they illustrate that the trained model can be adapted cross-lingually to an unseen language using only 25% of the target data. The authors emphasize the importance of training on multiple languages for very low-resource cross-lingual target scenarios but note that it is not crucial for multi-lingual testing scenarios. In such cases, they suggest the inclusion of large, well-prepared datasets for better performance.

Cho et al. (2018) utilize data from 10 BABEL languages to construct a multilingual seq2seq model as a prior model. They then employ transfer learning to adapt this model to four other BABEL languages. Various architectures are explored to enhance the prior multilingual seq2seq model. They have observed that the multi-lingual source models outperform the mono-lingual source models through experimentation.

Babu et al. (2021) introduces XLS-R, a large-scale model designed for cross-lingual speech representation learning, built upon the wav2vec 2.0 architecture. The authors train models with up to 2 billion parameters, utilizing nearly half a million hours of publicly available speech audio across 128 languages. This dataset represents a significant increase in scale compared to previous efforts in the field. Their evaluation encompasses a diverse set of tasks, domains, data regimes, and languages, spanning both high and low-resource scenarios. For speech recognition tasks, XLS-R surpasses the performance of previous state-of-the-art models on various datasets such as BABEL, MLS, CommonVoice, and VoxPopuli, with error rate reductions ranging from 14% to 34% relative on average. Overall, the authors anticipate that XLS-R will contribute to the improvement of speech processing tasks across a wide range of languages globally, thereby enabling advancements in multilingual speech technology.

2.4 Conclusion

In conclusion, although multilingual transfer learning models have shown significant advancements in ASR, they have yet to be thoroughly explored for the Sinhala language. This presents a clear research gap, as most existing work has focused on high-resource languages. My research aims to fill this gap by experimenting with Sinhala ASR using multilingual models for the first time, alongside techniques such as data augmentation and post-processing. These methods will contribute to developing more accurate and efficient ASR systems for Sinhala, advancing the state of speech recognition for this underrepresented language.

Chapter 3

Design

The primary objective of this research is to evaluate the effectiveness of transfer learning in Sinhala Automatic Speech Recognition (ASR). The goal is to utilize pre-trained models, referred to as source models, from languages with high amount of resources and adapt them for the Sinhala language. The XLS-R model developed by Babu et al. (2021), based on the wav2vec 2.0 framework by Baevski et al. (2020) has been selected as the source model for this research due to its high-resource nature and the fact that it has been proven to be effective by the authors. Reasons for this choice is explained further in the section 4.2. Figure 3.1 illustrates the pipeline of the ASR system developed in this research.

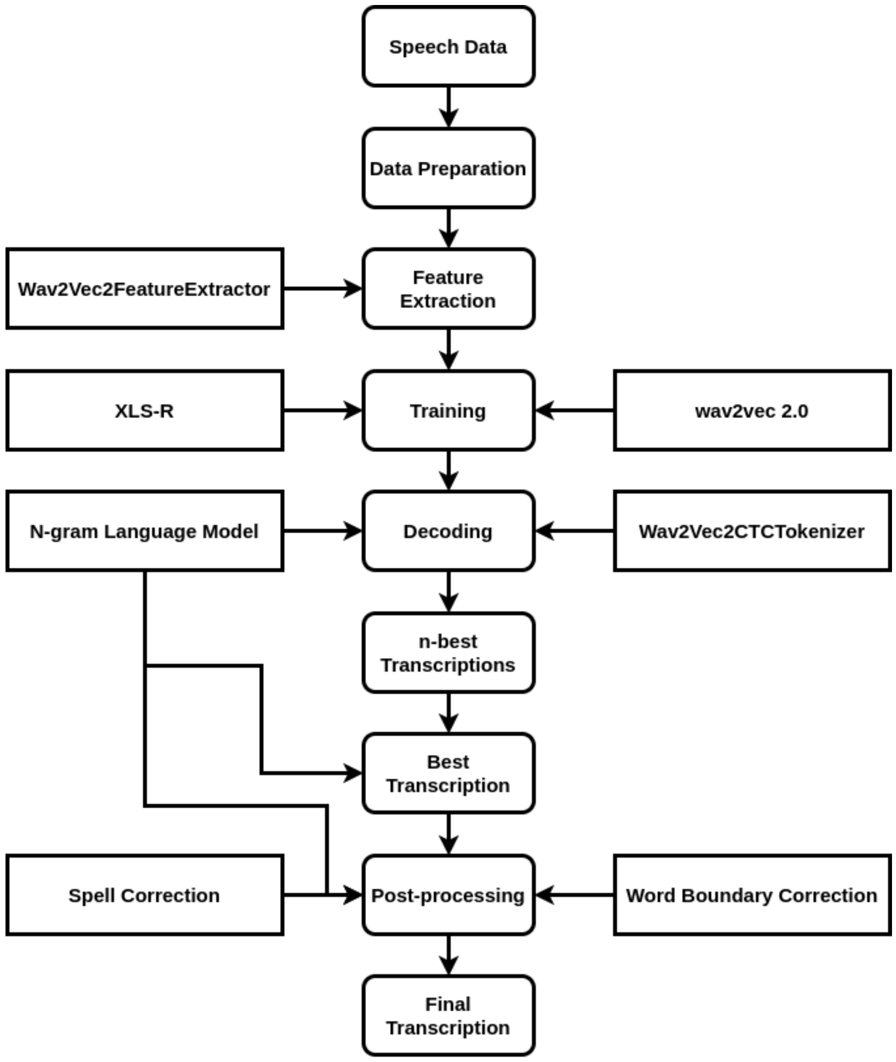


Figure 3.1: Pipeline of the research design

This section will delve deep into the XLS-R model which is built on wav2vec 2.0 (Baevski et al. 2020), and the inner workings behind the post-processing phase.

3.1 XLS-R

XLS-R is a large-scale model for cross-lingual speech representation learning based on wav2vec 2.0. It was trained using 128 languages with 436,000 hours of speech data including 54 hours of Sinhala speech data (Babu et al. 2021). The XLS-R model architecture is based on wav2vec 2.0, which is a framework for self-supervised learning of speech representations. They have employed a technique that entails analyzing spoken audio using a multi-layer convolutional neural network (CNN). Afterwards, they have employed masking on certain portions of the resulting latent speech representations, similar to the technique of masked language modeling (Devlin et al. 2018). The latent representations are subjected to additional processing using a Transformer network in order to provide contextualized representations. The training of the model involves a contrastive task, where the objective is to distinguish between the genuine latent representation and other irrelevant options (Baevski et al. 2020). Figure 3.2 shows an illustration of the wav2vec 2.0 model architecture.

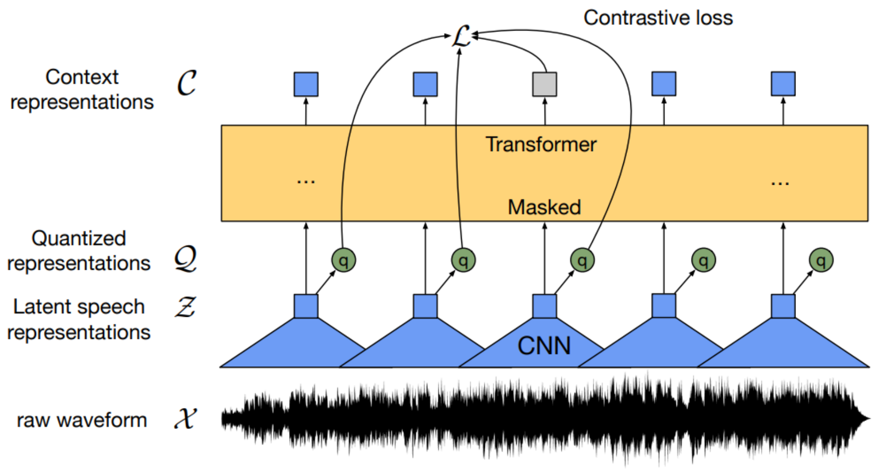


Figure 3.2: Illustration of wav2vec 2.0 (Baevski et al. 2020)

As mentioned earlier, the wav2vec 2.0 model consists of two main components: a multi-layer convolutional feature encoder, denoted as $f : X \rightarrow Z$, which takes raw audio input X and produces latent speech representations $z_1, ..., z_T$ for each time step T; and a Transformer, denoted as $g : Z \rightarrow C$, which processes these representations to create contextual embeddings $c_1, ..., c_T$ capturing information from the entire sequence. The output of the feature encoder is discretized to q_t using a quantization module $Z \rightarrow Q$ to represent the targets for the self-supervised objective. Each z_t corresponds to a 25ms segment of audio that is spaced 20ms apart, and the Transformer architecture used is based on Bidirectional Encoder Representations from Transformers (BERT) (Vaswani et al. 2017; Devlin et al. 2018; Babu et al. 2021).

During the training process, the representations generated by the feature encoder are transformed into discrete values $q_1, \ldots, q_T$ using a quantization module denoted as $Z \rightarrow Q$. This module is responsible for representing the targets in the objective function. The quantization process involves utilizing a Gumbel softmax to select entries from the codebooks, and these selected entries are then concatenated to form q (Jang et al. 2016; Jegou et al. 2010; Babu et al. 2021).

The model undergoes training by addressing a contrastive task involving masking spans of ten time steps with randomly chosen starting points. The goal is to correctly identify the true quantized latent representation for a masked time-step among a set of 100 distractors sampled from other masked time steps.

$$-\log\left(\frac{\exp(\text{sim}(c_t,q_t))}{\sum_{\tilde{q}\sim Q_t}\exp(\text{sim}(c_t,\tilde{q}))}\right)$$

Where:

- c_t is the output of the Transformer for time-step t.
- q_t is the true quantized latent representation for the masked time-step.
- Q_t is the set of $K = 100$ distractors sampled from other masked time steps.
- $\text{sim}(a, b)$ denotes the cosine similarity between vectors.
- $\tilde{q}$ represents each distractor sampled from Q_t.

The model is trained on various languages to acquire representations that work across different languages. During training, batches include samples from multiple languages, selected based on a distribution that considers the amount of unlabeled data available for each language. This distribution is controlled by an up-sampling factor, α, which balances the representation quality between languages with abundant data and those with limited resources (Babu et al. 2021).

To fine-tune the XLS-R model for Sinhala ASR, the approach outlined by Baevski et al. (2020) has been used. It involves appending a linear layer onto the pre-trained model to predict the output vocabulary. Connectionist Temporal Classification (CTC) is used for training (Graves et al. 2006).

Considering the settings suggested by Baevski et al. (2020), and the blog article[1] written by one of the authors of Babu et al. (2021), a learning rate of $3e-4$ which warms up for 500 steps and decays in a constant rate was used in this research. Figure 3.3 shows the change of the learning rate.

[1]`https://huggingface.co/blog/fine-tune-xlsr-wav2vec2`

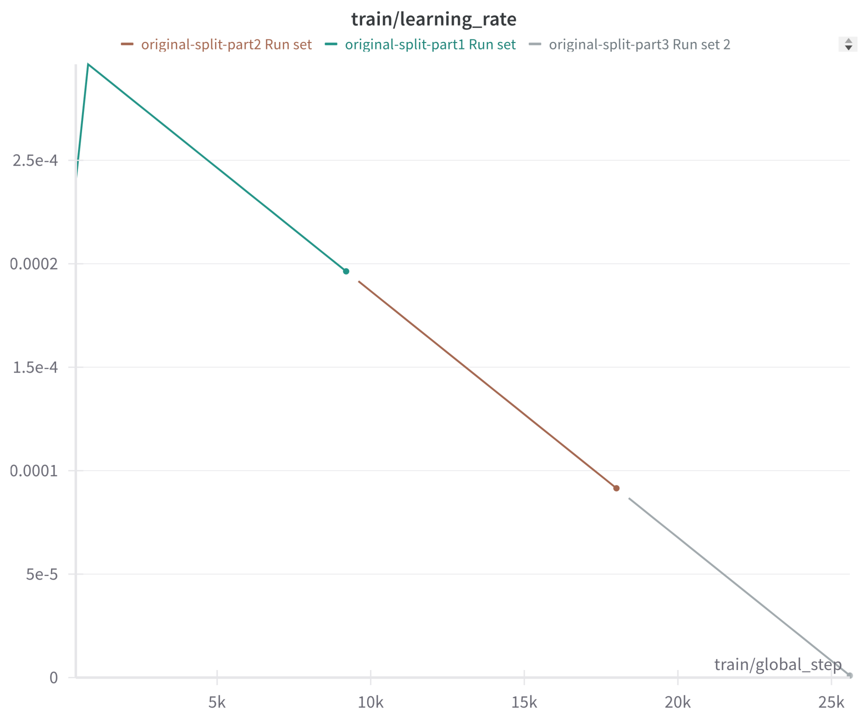


Figure 3.3: Graph illustrating the evolution of learning rate over the course of training

Batch sizes were selected according to the available resources.

3.2 Post-Processing

Post-processing is a step included into the pipeline of the research in order to mitigate two kinds of errors which could occur in the transcriptions of the ASR model.

- Spelling errors
- Word boundary errors, i.e., mistakes that occur due to incorrect separation of words within a sentence. For an example “සමහරවිට” vs “සමහර විට”.

3.2.1 Spelling Errors

First, in order to detect a spelling mistake, a language model (LM) is used. If the LM does not contain such word, it is considered as a spelling mistake. Then, a set of words which are closest to the incorrectly spelled word is selected from a the vocabulary of the LM. Levenshtein distance has been used to measure the closeness between two words. Then, a set of alternative transcriptions are created by replacing the incorrectly spelled word with the list of close words. The transcription which has the best LM score is returned at the end. Algorithm 1 has been used to correct the spelling mistakes in the transcriptions.

Algorithm 1 Spell Correction Algorithm

```
1: procedure CorrectSpellings($transcriptions$, $word\_list$)
2:     for $i \leftarrow 0$ to len($transcriptions$) $- 1$ do
3:         $best\_list \leftarrow$ empty_list
4:         $best\_list.append(transcriptions[i])$
5:         $words \leftarrow$ split $transcriptions[i]$ by spaces
6:         for $j \leftarrow 0$ to len($words$) $- 1$ do
7:             if $\text{lm.score}(words[j], \text{bos} = \text{False}, \text{eos} = \text{False}) \leq -5.9$ then
8:                 $closest\_words \leftarrow \text{get\_closest\_words}(words[j], word\_list)$
9:                 for $cw$ in $closest\_words$ do
10:                    $words[j] \leftarrow cw$
11:                    $best\_list.append(\text{join}(words))$
12:                end for
13:            end if
14:        end for
15:        if best_list is not empty then
16:            $transcriptions[i] \leftarrow \text{best\_t}(best\_list)$
17:        end if
18:    end for
19: end procedure
20: procedure get_closest_words($input\_word$, $word\_list$)
21:     $closest\_words \leftarrow$ list of words from $word\_list$ with Levenshtein distance
        $\leq 3$ from $input\_word$
22:     return $closest\_words$
23: end procedure
24: procedure best_t($t\_list$)
25:     $best\_score \leftarrow \text{lm.score}(t\_list[0])$
26:     $best\_transcription \leftarrow t\_list[0]$
27:     for $t$ in $t\_list$ do
28:         if $\text{lm.score}(t) \geq best\_score$ then
29:             $best\_score \leftarrow \text{lm.score}(t)$
30:             $best\_transcription \leftarrow t$
31:         end if
32:     end for
33:     return $best\_transcription$
34: end procedure
```

Levenshtein distance is used to calculate the similarity between two strings where the amount being 0 means the two strings are the same while higher values indicate increasing dissimilarity or the number of edits needed to transform one string into the other. Algorithm 2 shows the procedure to calculate the Levenshtein distance between two strings (Haldar and Mukhopadhyay 2011).

Algorithm 2 Calculate Levenshtein Distance

```
 1: procedure LevenshteinDistance($s$, $t$)
 2:     Set $n$ to be the length of $s$, set $m$ to be the length of $t$.
 3:     Construct a matrix $d$ containing 0..$m$ rows and 0..$n$ columns.
 4:     Initialize the first row to 0..$n$.
 5:     Initialize the first column to 0..$m$.
 6:     for $i = 1$ to $n$ do
 7:         for $j = 1$ to $m$ do
 8:             if $s[i] = t[j]$ then
 9:                 $cost \leftarrow 0$
10:             else
11:                 $cost \leftarrow 1$
12:             end if
13:             $d[i,j] \leftarrow \min(d[i-1,j]+1, d[i,j-1]+1, d[i-1,j-1]+\text{cost})$
14:         end for
15:     end for
16:     return $d[n,m]$
17: end procedure
```

3.2.2 Word Boundary Errors

Word boundary errors are resolved by iteratively splitting the words in a transcription morphologically and replacing the old single word by the new split words and checking the LM score to see whether it is transcription or not. Algorithm 3 shows the way it is done.

Algorithm 3 Word Boundary Correction Algorithm

```
1: procedure CorrectWordBoundary($transcriptions$, $word\_splitter$, $lm$)
2:     for $i \leftarrow 0$ to $\text{len}(transcriptions) - 1$ do
3:         $best\_list \leftarrow$ empty list
4:         $words \leftarrow$ split $transcriptions[i]$ by spaces
5:         for $j \leftarrow 0$ to $\text{len}(words) - 1$ do
6:             if $\text{len}(words[j]) \leq 1$ then
7:                 continue
8:             end if
9:             $res \leftarrow word\_splitter.split(words[j])$
10:            if $\text{lm.score}(res['base'], \text{bos} = \text{False}, \text{eos} = \text{False}) \leq -5.9$ or $\text{lm.score}(res['affix'], \text{bos} = \text{False}, \text{eos} = \text{False}) \leq -5.9$ then
11:                continue
12:            end if
13:            $new\_w \leftarrow res['base'] + "" + res['affix']$
14:            $checking \leftarrow$ join $words$ with spaces
15:            $words[j] \leftarrow new\_w$
16:            if $\text{lm.score}(checking) \leq \text{lm.score}(\text{join}(words))$ then
17:                append $\text{join}(words)$ to $best\_list$
18:            else
19:                $words[j] \leftarrow w$
20:            end if
21:        end for
22:        if $best\_list$ is not empty then
23:            $transcriptions[i] \leftarrow \text{best\_t}(best\_list)$
24:        end if
25:    end for
26: end procedure
```

Chapter 4

Implementation

The forthcoming sections will discuss the implementations conducted in each stage step by step.

4.1 Implementing the Baseline Models

The main baseline model considered in this research is the mono-lingual transfer learning model developed by Nanayakkara and Weerasinghe (2023). They have managed to achieve 22.93% WER (Word Error Rate) in their basic transfer learning model. And they have used data augmentation techniques to further reduce the WER to 17.19%. Furthermore, Nanayakkara and Weerasinghe (2023) has recreated the end-to-end model developed by Gamage, Pushpananda, Nadungodage, et al. (2021) as a baseline model. Since comparing with both a mono-lingual and an end-to-end model would increase the validity of the research, the end-to-end model by Gamage, Pushpananda, Nadungodage, et al. (2021) is recreated in this research as well.

All the baseline models are trained on Antpc server available at UCSC which is equipped with 4 Nvidia GeForce RTX 2080 Ti GPUs with each having a capacity of 10.8 GB. Efficient training process is achieved through training models using all 4 GPUs in a parallel manner.

4.1.1 Data Collection

Effective data preparation is crucial in ASR to maximize system performance and provide superior training outcomes. It also enhances the system's ability to handle various acoustic conditions, establishing a solid foundation for accurate speech recognition. Hence, a speech dataset is crucial.

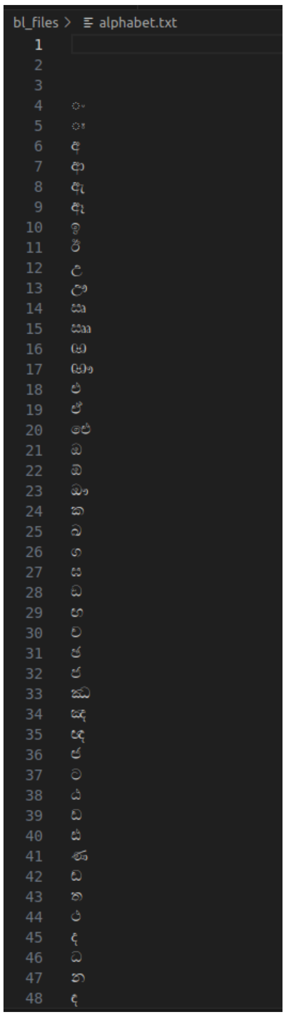
The speech dataset available at the Language Technology Research Laboratory (LTRL) of University of Colombo School of Computing (UCSC) which has 40 hours of training data which have been gathered using Praat (Boersma and Van Heuven 2001) and Redstart (Pammi et al. 2010) tools, is used in this research. A total of 45 individuals were recorded using Praat software, with 14 being male and 31 being female. There were a total of 78 individuals recorded for Redstart, with 23 of them being males and 55 being females. Each person has produced 200 utterances. The recordings from Praat were collected using a sample rate of 16kHz and in Redstart, the sample rate was 44.1kHz (Gamage, Pushpananda, Weerasinghe, et al. 2020). For testing, the dataset created by Gamage, Pushpananda, Weerasinghe, et al. (2020) which included recordings from 4 female speakers and 4 male speakers where they utter 80 speech sentences altogether is used.

4.1.2 Mono-lingual Transfer Learning Model

Mozilla's DeepSpeech tool is used to develop the mono-lingual transfer learning model, tracing the approach followed by Nanayakkara and Weerasinghe (2023). A pre-trained DeepSpeech model for English language which has been trained using the Librispeech dataset (Panayotov et al. 2015), has been used as a source model similar to the original research.

Data Preparation

According to the documentation provided by DeepSpeech, it is required to have a character alphabet of which ever language that the speech recognition system will be based on. Hence, it is required to create a text file (alphabet.txt) including all the characters in the Sinhala alphabet.



bl_files > alphabet.txt

```



ං
ඃ
අ
ආ
ඇ
ඈ
ඉ
ඊ
උ
ඌ
ඍ
ඎ
ඏ
ඐ
එ
ඒ
ඓ
ඔ
ඕ
ඖ
ක
ඛ
ග
ඝ
ඞ
ඟ
ච
ඡ
ජ
ඣ
ඤ
ඥ
ඦ
ට
ඨ
ඩ
ඪ
ණ
ඬ
ත
ථ
ද
ධ
න
ඳ
```

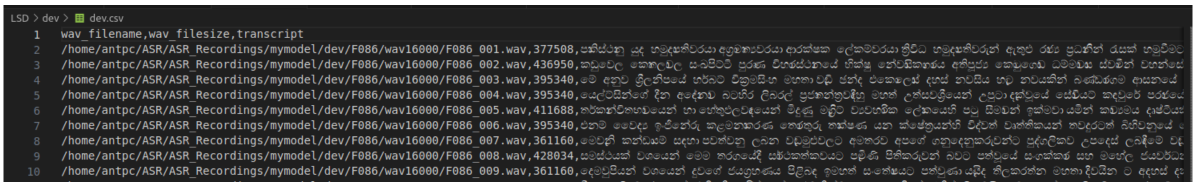
Figure 4.1: alphabet.txt file

All the transcriptions in the speech dataset should only contain characters from the alphabet. Failing to meet this condition will lead the model to crash. Hence, some kind of a filtration process is needed to eliminate any unrecognized characters from the transcriptions. Usually, when developing an English ASR, Zero-width space, Zero-width non-joiner, and Zero-width joiner characters are also removed. However, in Sinhala, those characters are necessary to be included in the alphabet for accurate transcription (Gamage, Pushpananda, Weerasinghe, et al. 2020).

The speech dataset is already split into 17848 utterances of training data and 2002 utterances of validation data. Hence, it is kept as it is. As discussed earlier, the testing dataset created by Gamage, Pushpananda, Weerasinghe, et al. (2020) is used as testing data.

For each of these splits, DeepSpeech requires creating files in the form of comma seperated values (CSV). Each CSV file must contain 3 columns.

- wav_filename : Contains the path to the audio file.
- wav_filesize : Contains the file size in bytes.
- transcript : Contains the transcript of the particular file.



```
LSD > dev > dev.csv
wav_filename,wav_filesize,transcript
/home/antpc/ASR/ASR_Recordings/mymodel/dev/F086/wav16000/F086_001.wav,377508,පාකිස්ථානු යුද හමුදාපතිවරයා අග්‍රාමාත්‍යවරයා ආරක්ෂක ලේකම්වරයා ත්‍රිවිධ හමුදාපතිවරුන් ඇතුළු රාජ්‍ය ප්‍රධානීන් රැසක් හමුවීමට
/home/antpc/ASR/ASR_Recordings/mymodel/dev/F086/wav16000/F086_002.wav,436950,කඩුවෙල කොතලාවල සංඝපිටිය පුරාණ විහාරස්ථානයේ භික්ෂු නේවාසිකාගාරය අතිපූජ්‍ය කොළුගොඩ ධම්මවාස ස්වාමීන් වහන්සේ
/home/antpc/ASR/ASR_Recordings/mymodel/dev/F086/wav16000/F086_003.wav,395340,මේ අනුව ශ්‍රීලනිපයේ හර්බට් වික්‍රමසිංහ මහතා වැඩි ඡන්ද එකොළොස් දහස් නවසිය හැට නවයකින් බණ්ඩාරගම ආසනයේ
/home/antpc/ASR/ASR_Recordings/mymodel/dev/F086/wav16000/F086_004.wav,395340,යෙල්ට්සින්ගේ දින අදේනාව බටහිර ලිබරල් ප්‍රජාතන්ත්‍රවාදීහු මහත් උත්සවශ්‍රීයෙන් උපුටා දැක්වූයේ සෝවියට් කඳවුරේ පරාජයේ
/home/antpc/ASR/ASR_Recordings/mymodel/dev/F086/wav16000/F086_005.wav,411688,තර්කාන්විතභාවයෙන් හා හේතුඵලවාදයෙන් මිදුණු මශ්‍රිට ව්‍යවහාරික ලෝකයෙහි පටු සීමාවන් ඉක්මවා යමින් කාව්‍යමය දෘෂ්ටිය
/home/antpc/ASR/ASR_Recordings/mymodel/dev/F086/wav16000/F086_006.wav,395340,එනම් වෛද්‍ය ඉංජිනේරු කළමනාකරණ තොරතුරු තාක්ෂණ යන ක්ෂේත්‍රයන්හි විද්වත් වෘත්තිකයන් තවදුරටත් බිහිවනුයේ
/home/antpc/ASR/ASR_Recordings/mymodel/dev/F086/wav16000/F086_007.wav,361160,මෙවැනි කණ්ඩායම් සඳහා පවත්වනු ලබන වැඩමුළුවලට අමතරව අපගේ ගනුදෙනුකරුවන්ට පුද්ගලිකව උපදෙස් ලබාදීමේ
/home/antpc/ASR/ASR_Recordings/mymodel/dev/F086/wav16000/F086_008.wav,428034,සමස්ථයක් වශයෙන් මෙම තරගයේදී සාර්ථකත්වයට පැමිණි පිතිකරුවන් බවට පත්වූයේ සංගක්කාර සහ මහේල
/home/antpc/ASR/ASR_Recordings/mymodel/dev/F086/wav16000/F086_009.wav,361160,දෙමව්පියන් වශයෙන් දුවගේ ජයග්‍රහණය පිළිබඳ ඉමහත් සංතෝෂයට පත්වුණා යයිද තිලකරත්න මහතා දිවයින ට අදහස්
```

Figure 4.2: Example CSV file (dev.csv)

Building the Language Model

The integration of a language model into DeepSpeech ASR improves transcription accuracy by utilizing contextual predictions. The language model will enhance the output transcriptions generated by the ASR system using techniques such as correcting erroneous spellings, addressing incorrect word boundaries, and managing ambiguous words. In DeepSpeech documentation, language models are also known as external scorers. KenLM toolkit is used in DeepSpeech to create the n-gram language models (Heafield 2011).

A text corpus is needed to create the language model. The text corpus used by Gamage, Pushpananda, Nadungodage, et al. (2021) to create the language model is used in this research as well. This corpus is a combination of 3 corpora, UCSC Novel Corpus with 90000 unique sentences, Chatbot Corpus with 388 unique sentences and the corpus created using active learning method which has 20000 unique sentences (Gamage, Pushpananda, Nadungodage, et al. 2021).

DeepSpeech has provided two python scripts for the creation of the external scorer. One is to create the trie type n-gram language model and the other one is to create the external scorer as a package. Following is the script to create the 4-gram language model.

```
python3 generate_lm.py \
--input_txt /hdd/2019CS125/corpus/corpus_20000+90000+chatbot.txt\
--output_dir /hdd/2019CS125/bl_files/lm_tl_base_model/lm/ \
--top_k 500000 \
--kenlm_bins /hdd/2019CS125/kenlm/build/bin \
--arpa_order 4 \
--max_arpa_memory "85\%" \
--arpa_prune "0|0|1" \
--binary_a_bits 255 \
--binary_q_bits 8 \
--binary_type trie
```

This script will create two files: lm.binary, which is the language model in binary format, and vocab-500000.txt, which is the vocabulary set of the corpus. The following script will wrap them in an external scorer package.

```
./generate_scorer_package \
--alphabet /hdd/2019CS125/bl_files/alphabet.txt \
--lm /hdd/2019CS125/bl_files/lm_tl_base_model/lm/lm.binary \
--vocab /hdd/2019CS125/bl_files/lm_tl_base_model/lm/vocab-500000.
   txt \
--package /hdd/2019CS125/bl_files/lm_tl_base_model/lm/kenlm.scorer
   \
--default_alpha 0.931289039105002 \
--default_beta 1.1834137581510284
```

lm_optimizer.py script provided by DeepSpeech can be used to find the best values for alpha and beta for Sinhala since the original values are for English. However, the original values used by Nanayakkara and Weerasinghe (2023) has been used here.

Training

In the DeepSpeech documentation, it is recommended to create a python virtual environment to do the training. A python virtual environment is created with the following command.

```
virtualenv -p python3 $HOME/tmp/deepspeech-venv/
```

Once the virtual environment is created, the following command will activate it.

```
source $HOME/tmp/deepspeech-venv/bin/activate
```

Although Nanayakkara and Weerasinghe (2023) has done several experimentation with different models, the best model has been selected to recreate, which is the mono-lingual transfer learning model with data augmentations applied. Following is the training script which is of similar settings to the original research.

```
#! /bin/sh

source /hdd/2019CS125/deepspeech-train-venv/bin/activate;

export CUDA_VISIBLE_DEVICES="2,3"

python /hdd/2019CS125/DeepSpeech-0.9.3/DeepSpeech.py \
--train_cudnn \
--drop_source_layers 1 \
--alphabet_config_path /hdd/2019CS125/bl_files/alphabet.txt \
--scorer /hdd/2019CS125/bl_files/lm_tl_base_model/lm/kenlm.scorer \
--learning_rate 0.00095 \
--reduce_lr_on_plateau True \
--plateau_epochs 8 \
--plateau_reduction 0.08 \
--early_stop True \
--dropout_rate 0.22 \
--save_checkpoint_dir /hdd/2019CS125/baseline_tl_models/
   tl3_DA_mono_DS/checkpoints_2 \
--load_checkpoint_dir /hdd/2019CS125/data/deepspeech-pretrained/
   deepspeech-0.9.3-checkpoint \
--train_files /hdd/2019CS125/final_dataset/LSD/train/train.csv \
--dev_files /hdd/2019CS125/final_dataset/LSD/dev/dev.csv \
--test_files /hdd/2019CS125/final_dataset/LSD/test/test.csv \
--augment overlay[p=0.3,source=/hdd/2019CS125/final_dataset/LSD/
   train/noise.csv,layers=10:1,snr=50:20~9] \
--augment reverb[p=0.1,delay=50.0~30.0,decay=10.0:2.0~1.0] \
--augment codec[p=0.4,bitrate=48000:16000] \
--augment volume[p=0.4,dbfs=-10:-40] \
--augment pitch[p=0.4,pitch=1~0.2] \
--augment tempo[p=0.4,factor=1~0.5] \
--augment frequency_mask[p=0.4,n=1:3,size=1:5] \
--augment time_mask[p=0.4,domain=signal,n=3:10~2,size=50:100~40] \
--augment dropout[p=0.4,rate=0.05] \
--augment add[p=0.4,domain=signal,stddev=0~0.5] \
--augment multiply[p=0.4,domain=features,stddev=0~0.5] \
--use_allow_growth \
> /hdd/2019CS125/baseline_tl_models/tl3_DA_mono_DS/results_2.txt
   2>&1;
```

In the above script, as discussed earlier, an English, pre-trained, DeepSpeech model is used as the source model. "load_checkpoint_dir" flag is used to set the path to the source model. In DeepSpeech, when implementing mono-lingual transfer learning, after loading the weights from the source model, the top layer is dropped so that the model can learn new weights for the target language data. "drop_source_layers" flag is used to drop the top layer (layer 1) (Nanayakkara and Weerasinghe 2023).

Plateau reduction is a mechanism used by Nanayakkara and Weerasinghe (2023) when developing the mono-lingual model to achieve a smooth training process. Plateau reduction is when it defines a number of epochs where if the loss doesn't get reduced for the said number of epochs it will adjust the learning rate as mentioned in the script.

"augment" flag has been used to apply different augmentations to the train set. The "p" parameter in each augmentation denotes the probability which an audio file will get augmented so that the whole train set doesn't get augmented.

The final line of the script is to redirect the output of the script to a text file which is optional.

Exporting the model for inference

Upon completion of the training process, a model file named output_graph.pb is produced. This file has additional loading time and consumes more memory. Producing a mmap-able model can avoid this. Following command can be used to create such a model.

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

Testing

Following script can be used to test the trained model using the test dataset.

```
#! /bin/sh

# source /hdd/2019CS125/deepspeech-train-venv/bin/activate;

export CUDA_VISIBLE_DEVICES="0,1,2,3"

python /hdd/2019CS125/DeepSpeech-0.9.3/DeepSpeech.py \
--show_progressbar True \
--train_cudnn \
--test_batch_size 8 \
--save_checkpoint_dir /hdd/2019CS125/baseline_tl_models/
   tl3_DA_mono_DS/checkpoints_2 \
--load_checkpoint_dir /hdd/2019CS125/baseline_tl_models/
   tl3_DA_mono_DS/checkpoints_2 \
--test_output_file results_my_2.json \
--alphabet_config_path /hdd/2019CS125/bl_files/alphabet.txt \
--scorer /hdd/2019CS125/bl_files/lm_tl_base_model/lm/kenlm.scorer \
--lm_alpha 0.7780193880793429 \
--lm_beta 4.901133686880696 \
--export_dir /hdd/2019CS125/baseline_tl_models/tl3_DA_mono_DS/
   exp_my \
--test_files /hdd/2019CS125/final_dataset/LSD/test/test.csv \
> /hdd/2019CS125/baseline_tl_models/tl3_DA_mono_DS/results_test_2.
   txt 2>&1;
```

Alpha and beta values are used as same as in the original research (Nanayakkara and Weerasinghe 2023).

4.1.3 End-to-end Model

As mentioned earlier, in order to compare the multi-lingual transfer learning model, both the mono-lingual model developed by Nanayakkara and Weerasinghe (2023), and the end-to-end model created by Gamage, Pushpananda, Nadungodage, et al. (2021) will be recreated in this research as baseline models. However, rather than recreating the exact model developed by Gamage, Pushpananda, Nadungodage, et al. (2021), the version which was developed by Nanayakkara and Weerasinghe (2023) as a baseline model for their research will be implemented as they have managed to get a slightly better accuracy using plateau handling.

The first stages of developing the end-to-end model is quite similar to developing the mono-lingual transfer learning model since this is also developed using DeepSpeech engine. Thus, after preparing the alphabet, language model, and the CSV files as discussed in the previous section, the following script is used to train the model together with plateau handling.

```
python -u DeepSpeech.py
--train_files /hdd/2019CS125/Test1/ASRDATA/mymodel/train/train.csv
--dev_files /hdd/2019CS125/Test1/ASRDATA/mymodel/dev/dev.csv
--test_files /hdd/2019CS125/Test1/ASRDATA/mymodel/test/test.csv
--train_batch_size 100
--dev_batch_size 60
--test_batch_size 40
--n_hidden 375
--epochs 100
--learning_rate 0.00095
--reduce_lr_on_plateau True
--plateau_epochs 8
--plateau_reduction 0.08
--early_stop True
--dropout_rate 0.22
--report_count 100
```

```
--export_dir /hdd/2019CS125/baseline_tl_models/test1_e2e_plateau/
   exp/
--checkpoint_dir /hdd/2019CS125/baseline_tl_models/
   test1_e2e_plateau/checkpoints/
--alphabet_config_path /hdd/2019CS125/Test1/sin/sinhala_alphabet.
   txt
--scorer /hdd/2019CS125/bl_files/lm_tl_base_model/lm/kenlm.scorer >
    /hdd/2019CS125/baseline_tl_models/test1_e2e_plateau/results.txt
    2>&1
```

4.2 Implementing the Multi-lingual Transfer Learning Model

The initial decision was to conduct experiments using three different combinations of source languages. Languages that are related to Sinhala, high-resourced languages, and a mix of the two. However, it is challenging to find pre-trained models that have been trained using these specific language combinations. Particularly, models that are trained exclusively on languages related to Sinhala. Additionally, training a multi-lingual source model from the scratch is challenging due to the constraints imposed by limited resources. Therefore, following an extensive investigation, the options were reduced to two distinct models. XLSR-53 model (Conneau et al. 2020) and the XLS-R model (Babu et al. 2021) which are developed by Facebook AI.

XLSR-53 model is trained using 53 languages which includes both high and low resource languages with 56,000 hours of speech data (Conneau et al. 2020). Compared to the XLSR-53 model, according to Babu et al. (2021), XLS-R model is an updated and more powerful version which has been trained using 128 languages with 436,000 hours of speech data. In addition, they have utilized 54 hours of Sinhala speech data which is a bit larger than the one that will be using in this research as well. Hence, it was evident that the preferable choice would be the XLS-R model and the results obtained by Babu et al. (2021) also validated it. XLS-R model has 3 different sizes. 300 million parameters, 1 billion parameters and 2 billion parameters. Due to the resource constraints, 300 million parameter version has been selected as the source model for constructing the multi-lingual transfer learning model.

Since the antpc server available at UCSC turned out to be inadequate in terms of memory requirements to train this model, Kaggle platform has been used. In Kaggle, an Nvidia Tesla P100 GPU has been used for the training process.

4.2.1 Data Preparation

First of all, the Sinhala dataset which will be using to fine-tune the XLS-R model should be uploaded to Kaggle. The same dataset used to implement the baseline models, the 40 hour speech dataset available at the LTRL of the UCSC, will be used.

After creating a dataset in Kaggle, the next task is to set up a notebook and adding the created dataset into the notebook so that we can load the data and perform the necessary pre-processing. In order to load the data, the “Datasets” library of Hugging Face is used. Hugging Face is a community which provides many resources such as libraries, models and datasets for training various machine learning models.

The “load_dataset” method of the “Datasets” library can be used to load the dataset using a CSV file. The CSV file format is only slightly different from the one used in DeepSpeech when implementing the baseline models which is shown in the figure 4.2. The CSV file format is not strict unlike in DeepSpeech so that the columns names and additional data can be kept as desired. Following are the columns used in the CSV file.

- path: Contains the path of the audio file
- sentence: Contains the transcription of the audio file
- audio: Contains the path of the audio file

The path of the audio file is duplicated in “path” and “audio” columns since it will be later replaced by the decoded audio data in the “audio” column. Following code is used to load the dataset using the CSV files for train and validation (eval) sets.

```
sinhala_data = load_dataset(
    'csv', data_files={
        'train': '/kaggle/input/sinhala-asr-data/train_kaggle1_full.
           csv',
        'eval': '/kaggle/input/sinhala-asr-data/eval_kaggle_new.csv'
    }
)
```

As mentioned earlier, audio data is required to be decoded and represented as a 1-dimensional array in order to use in the pre-processing stage later. In order to do this, the “Features” class of the “Datasets” library can be used. The “Features” class is used to define the attributes of the dataset. When the in-built “Audio” feature is defined, the audio file will be automatically decoded and resamples if the sampling rate is mentioned. Following shows the code to defining the features and casting them on to the loaded dataset.

```
features = Features(
    {
        "path": Value("string"),
        "sentence": Value("string"),
        "audio": Audio(sampling_rate=16000),
    })

sinhala_data = sinhala_data.cast(features)
```

The next step is to remove any special characters from the transcriptions. It is considerably harder to assign speech chunks to such special characters because they don't actually correspond to a typical sound unit. The following function was designed to remove the special characters using regular expressions.

```
chars_to_remove_regex = '[\)…"‘" \(\\,\?\.\!\-\;\:\"\\%\\\\']'

def remove_special_characters(batch):
    batch["sentence"] = re.sub(chars_to_remove_regex, '', batch["
        sentence"])
    return batch
```

Then the above function is mapped to the loaded datasets using the following code.

```
sinhala_data["train"] = sinhala_data["train"].map(
    remove_special_characters)
sinhala_data["eval"] = sinhala_data["eval"].map(
    remove_special_characters)
```

XLS-R is fine-tuned using Connectionist Temporal Classification (CTC) algorithm. In CTC, the speech chunks are usually classified into characters. Hence, it is required to have the set of unique characters in the transcriptions in both the train and validation sets. In order to perform this task, first, the set of transcriptions are joined into a single string and all the unique characters are extracted in both train and validation sets using the following code.

```
def extract_all_chars(batch):
  all_text = "␣".join(batch["sentence"])
  vocab = list(set(all_text))
  return {"vocab": [vocab], "all_text": [all_text]}

vocab_train = sinhala_data["train"].map(extract_all_chars, batched=
    True, batch_size=-1, keep_in_memory=True, remove_columns=
    sinhala_data["train"].column_names)

vocab_eval = sinhala_data["eval"].map(extract_all_chars, batched=
    True, batch_size=-1, keep_in_memory=True, remove_columns=
    sinhala_data["eval"].column_names)
```

Then, the union of two sets of characters extracted from train and validation sets is acquired and converted into a sorted, enumerated dictionary as follows.

```
vocab_list = list(set(vocab_train["vocab"][0]) | set(vocab_eval["
   vocab"][0]))

vocab_dict = {v: k for k, v in enumerate(sorted(vocab_list))}
```

In order to make it clearer that " "(white space) has its own token class, a more visible character, "|" is replaced. An "unknown" token should be added to deal with the previously unseen characters that might appear. In addition, the CTC algorithm requires the inclusion of a padding token that corresponds to the "blank token" of the CTC algorithm, which is an essential element of the algorithm. Following code performs the above-mentioned tasks.

```
vocab_dict["|"] = vocab_dict["␣"]
del vocab_dict["␣"]

vocab_dict["[UNK]"] = len(vocab_dict)

vocab_dict["[PAD]"] = len(vocab_dict)
```

And then, the extracted vocabulary should be saved as a json file.

```
with open('vocab.json', 'w') as vocab_file:
    json.dump(vocab_dict, vocab_file, ensure_ascii=False)
```

4.2.2 Creating Tokenizer and Feature Extractor

Since ASR models transcribe the speech signal to text, a feature extractor is required to convert the speech signals into feature vectors in order to be compatible with the input format of the model. Additionally, a tokenizer is required to process the output of the model to text format. The pre-trained XLS-R model is trained using the wav2vec 2.0 framework (Baevski et al. 2020). Hence, the tokenizer and the feature extractor provided by the Hugging Face's Transformers library, Wav2Vec2CTCTokenizer and Wav2Vec2FeatureExtractor can be used.

An instance of the Wav2Vec2CTCTokenizer class should be created and the previously created vocab.json file should be used to load the vocabulary into the instance.

```
tokenizer = Wav2Vec2CTCTokenizer.from_pretrained("/kaggle/working/"
    , unk_token="[UNK]", pad_token="[PAD]", word_delimiter_token="|"
    )
```

It is best to save the created tokenizer in the Hugging Face Hub which is a github like version control system for ML models.

The following code will create the Wav2Vec2FeatureExtractor object.

```
feature_extractor = Wav2Vec2FeatureExtractor(feature_size=1,
    sampling_rate=16000, padding_value=0.0, do_normalize=True,
    return_attention_mask=True)
```

The purposes of the parameters of the Wav2Vec2FeatureExtractor are as follows.

feature_size: This parameter specifies the size of each feature vector in the input sequence. For Wav2Vec2, the model was trained on raw speech signals, so each feature vector represents a specific aspect of the speech signal. In the case of Wav2Vec2, the feature size is typically set to 1 because the model was trained on raw speech signals directly.

sampling_rate: This parameter indicates the sampling rate at which the model was trained. The sampling rate represents how many samples of the audio signal were taken per second during training. It's crucial to set this parameter correctly to match the sampling rate of the input audio data during inference.

padding_value: During batched inference, inputs may have different lengths. To handle this variability, shorter inputs are typically padded with a specific value. This parameter specifies the value to use for padding.

do_normalize: This is a boolean parameter that determines whether the input should be normalized or not before feeding it into the model. Normalization typically involves zero-mean-unit-variance normalization, where the input is scaled to have zero mean and unit variance. Speech models often perform better when the input is normalized.

return_attention_mask: This parameter specifies whether the model should utilize an attention mask during batched inference. An attention mask is used to indicate which elements of the input sequence should be attended to and which should be ignored. For XLS-R models, it's generally recommended to use an attention mask.

For the ease of use, the tokenizer and the feature extractor are wrapped into a single processor class.

```
processor = Wav2Vec2Processor(feature_extractor=feature_extractor,
    tokenizer=tokenizer)
```

4.2.3 Data Pre-processing

In the data preparation step, the audio files were decoded and represented as 1-dimensional arrays. Now, those data should be converted into a format which will be compatible for input of the Wav2Vec2ForCTC model for training.

First, the content of the 1-dimensional array of each audio file is extracted and normalized by the Wav2Vec2Processor. Then, the transcriptions are encoded into label ids. These tasks are done using the following function.

```python
def prepare_dataset(batch):
    audio = batch["audio"]

    batch["input_values"] = processor(audio["array"], sampling_rate
       =audio["sampling_rate"]).input_values[0]
    batch["input_length"] = len(batch["input_values"])

    with processor.as_target_processor():
        batch["labels"] = processor(batch["sentence"]).input_ids
    return batch

sinhala_data["train"] = sinhala_data["train"].map(prepare_dataset,
    remove_columns=sinhala_data["train"].column_names)

sinhala_data["eval"] = sinhala_data["eval"].map(prepare_dataset,
    remove_columns=sinhala_data["eval"].column_names)
```

XLS-R typically has a greater input length than an output length. Therefore, it is advisable to pad the input batches in a way that each input of a batch is padded to the length of the highest input length of that particular batch rather than padding all inputs to the highest length of the whole dataset. In order to perform this, a data collator needs to be defined. Hugging Face's Transformers library has an example in their github repository. It is as follows.

```python
@dataclass
class DataCollatorCTCWithPadding:
    processor: Wav2Vec2Processor
    padding: Union[bool, str] = True

    def __call__(self, features: List[Dict[str, Union[List[int],
        torch.Tensor]]]) -> Dict[str, torch.Tensor]:

        input_features = [{"input_values": feature["input_values"]}
            for feature in features]
        label_features = [{"input_ids": feature["labels"]} for
            feature in features]

        batch = self.processor.pad(
            input_features,
            padding=self.padding,
            return_tensors="pt",
        )
        with self.processor.as_target_processor():
            labels_batch = self.processor.pad(
                label_features,
                padding=self.padding,
                return_tensors="pt",
            )
        labels = labels_batch["input_ids"].masked_fill(labels_batch.
            attention_mask.ne(1), -100)

        batch["labels"] = labels

        return batch

data_collator = DataCollatorCTCWithPadding(processor=processor,
    padding=True)
```

4.2.4 Training

First of all, an evaluation metric to evaluate the model should be defined. The “load_metric” method of the “Datasets” library can be used for this task. A function to calculate the metric should be defined as well. As usual, the Word Error Rate (WER) is chosen as the evaluation metric. To calculate the WER, the argmax of the prediction logits is selected since it will be the most likely transcription in the form of token ids. Then the padded tokens are replaced. The data collator introduced -100 as the padding token. It will be replaced by the padding token of the tokenizer. Then, predictions ids and the label ids are decoded and the compute function is called for the list of predictions and the references to calculate the WER.

```python
wer_metric = load_metric("wer")

def compute_metrics(pred):
    pred_logits = pred.predictions
    pred_ids = np.argmax(pred_logits, axis=-1)

    pred.label_ids[pred.label_ids == -100] = processor.tokenizer.
        pad_token_id

    pred_str = processor.batch_decode(pred_ids)

    label_str = processor.batch_decode(pred.label_ids, group_tokens
       =False)

    wer = wer_metric.compute(predictions=pred_str, references=
       label_str)

    return {"wer": wer}
```

The pre-trained source model should be loaded now. As discussed, the 300 million parameter version of the XLS-R is loaded as follows.

```
model = Wav2Vec2ForCTC.from_pretrained(
    "facebook/wav2vec2-xls-r-300m",
    attention_dropout=0.0,
    hidden_dropout=0.0,
    feat_proj_dropout=0.0,
    mask_time_prob=0.05,
    layerdrop=0.0,
    ctc_loss_reduction="mean",
    pad_token_id=processor.tokenizer.pad_token_id,
    vocab_size=len(processor.tokenizer),
)
```

When fine-tuning a pre-trained model, the layers of the already trained model is freezed to prevent it from training again since it has been trained enough. A linear layer is added on top of the pre-trained model to train the model on the labeled data of the target language. The “freeze_feature_extractor()” function is used to freeze the pre-trained model.

Finally, the training arguments should be defined using the “TrainingArguments” class of the Transformers library and the trainer should be defined using the ”Trainer” class.

```
model.freeze_feature_extractor()

training_args = TrainingArguments(
  output_dir=repo_name,
  group_by_length=True,
  hub_strategy="checkpoint",
  per_device_train_batch_size=8,
  gradient_accumulation_steps=2,
  evaluation_strategy="steps",
  num_train_epochs=30,
  gradient_checkpointing=True,
  save_steps=400,
  eval_steps=400,
  logging_steps=400,
  learning_rate=3e-4,
  warmup_steps=500,
  save_total_limit=2,
  push_to_hub=True,
)

trainer = Trainer(
    model=model,
    data_collator=data_collator,
    args=training_args,
    compute_metrics=compute_metrics,
    train_dataset=sinhala_data["train"],
    eval_dataset=sinhala_data["eval"],
    tokenizer=processor.feature_extractor,
)
```

To begin the training, “train()” function has to be called. It is advisable to save the trained model to the Hugging Face Hub in order to easily perform testing and other alterations such as integrating a language model(LM).

4.2.5 Language Model Integration

Typically, ASR models require the use of an additional language model (LM) and a dictionary in order to convert the sequence of classified audio frames into a coherent transcription. The architecture of Wav2Vec2 is built upon transformer layers, allowing each processed audio representation to have contextual information from all other audio representations. Furthermore, Wav2Vec2 utilizes the CTC algorithm during the fine-tuning process to address the challenge of aligning the difference of length between the input audio and the output text (Baevski et al. 2020). Thus, it is optional to have an LM to decode the ASR output of the multi-lingual transfer learning model.

However, using an LM has been proven to be beneficial in terms of accuracy and the readability of the transcriptions by Baevski et al. (2020). Kensho Technologies' pyctcdecode library has been used to integrate an LM to the trained model.

First of all, the KenLM toolkit (Heafield 2011) is used to create an n-gram LM. The “lmplz” command can be used to build the n-gram LM as follows. The corpus used to generate the LM is the same as the one used to generate the LM when implementing the baseline DeepSpeech models as explained in the sub section 4.1.2.

```
kenlm/build/bin/lmplz -o 5 < "corpus_20000+90000+chatbot.txt" > "5
    gram.arpa"
```

The next step is to combine the generated LM with the previously trained model. In order to do that, first, the trained model which was previously save to the Hugging Face Hub should be loaded using the “AutoProcessor” class provided by the Hugging Face Transformers as follows.

```
processor = AutoProcessor.from_pretrained("SpideyDLK/wav2vec2-large
    -xls-r-300m-sinhala-original-split-part4-epoch30-final")
```

Kensho Technologies’ pyctcdecode library has the method, “build_ctcdecoder” which requires the list of vocabulary of the tokenizer of the trained ASR model and the .arpa file generated previously. The list of vocabulary is extracted from the tokenizer of the loaded model and converted into a sorted list and passed to the “build _ctcdecoder” function with the path to the LM as follows.

```
vocab_dict = processor.tokenizer.get_vocab()
sorted_vocab_dict = {k: v for k, v in sorted(vocab_dict.items(),
    key=lambda item: item[1])}

decoder = build_ctcdecoder(
    labels=list(sorted_vocab_dict.keys()),
    kenlm_model_path="/hdd/2019CS125/final_lms/chatbot_lm/5gram.
        arpa"
)
```

Then, the “Wav2Vec2ProcessorWithLM” class can be used to wrap the feature extractor, processor, and the generated decoder together as follows.

```
processor_with_lm = Wav2Vec2ProcessorWithLM(
    feature_extractor=processor.feature_extractor,
    tokenizer=processor.tokenizer,
    decoder=decoder
)
```

Finally, the modified model with the integrated LM should be pushed back into the Hugging Face Hub to be used in testing.

4.2.6 Testing

Testing the created ASR model should be done with and without the LM in order to compare the outputs in each approach.

First of all, the testing dataset should be loaded into the notebook in a similar way explained in the section 4.2.1. Then, the set of 1-dimensional arrays which represent each audio file should be extracted into a list. It can be done as follows.

```
inputs_list = []

processor = Wav2Vec2Processor.from_pretrained("SpideyDLK/wav2vec2-
    large-xls-r-300m-sinhala-original-split-part4-epoch30-final")

for audio_sample in test_data["test"]:
    inputs = processor(audio_sample["audio"]["array"],
        sampling_rate=16_000, return_tensors="pt")
    inputs_list.append(inputs)
```

Then, the created list should be passed on to the trained model the results should be stored in a list as well.

```
logits_array = []

for input_data in inputs_list:
    with torch.no_grad():
        logits = model(**input_data).logits
        logits_array.append(logits)
```

Testing without the LM

From the list of logits received from the above step, the argmax is taken to get the set of token ids for the transcription for each audio file.

```
pred_id_array = []

for logit in logits_array:
    ids = torch.argmax(logit, dim=-1)[0]
    pred_id_array.append(ids)
```

Since the transcriptions are now in the form of token ids, it should be decoded into readable text. It is done as follows;

```
predictions_without_lm = []

for ids in pred_id_array:
    pred = processor.decode(ids)
    predictions_without_lm.append(pred)
```

Since the processor is loaded using the “Wav2Vec2Processor” class, the LM will not be loaded. Hence, the decoded transcription is without using an LM.

Testing with the LM

In order to decode with the LM, the processor should be loaded with the “Wav2Vec2ProcessorWithLM” class. Then the logits should be converted into numpy to be able to be compatible with the pyctcdecode library and passed on to the processor to decode. Following code shows the tasks above.

```
transcription_with_lm = []

with torch.no_grad():
    for logits in logits_array:
        transcription = processor.batch_decode(logits.numpy(),
                                              unk_score_offset=-1.0,
                                              beam_width=1000,
                                              beam_prune_logp=-20.0,
                                              alpha=0.9,
                                              beta=5.7,
                                              lm_score_boundary=False,
                                              output_word_offsets=False
                                             ).text
        transcription_with_lm.append(transcription[0])
```

The hyperparameters used in the above code can be obtained from a grid search algorithm to find out the best parameters which will provide the most accurate transcriptions.

Calculating the WER

Finally, the WER values should be calculated. “jiwer” is a library that can be used for this task. It requires the set of references to behave as the ground truths to calculate the WER. The set of references should be extracted from the loaded dataset and passed on to the “wer” function of the “jiwer” library together with the set of transcriptions which we need to include in the calculation.

```
from jiwer import wer

print(wer(sentences,transcription_with_lm)*100)
```

4.2.7 Post-processing

Even after integrating an LM to decode the outputs of the ASR model, there can be some types of errors in the transcriptions which will massively contribute to the WER. After analyzing the output received from the multi-lingual transfer learning model, the following prominent errors were identified.

- Spelling errors
  E.g. :

  වෙළඳාම vs වෙළෙඳාම
  හැඟිණ vs හැඟින
  අයියණ්ඩිගේ vs අයියන්ඩිගේ
  සිතිණ vs සිතින

- Words boundary errors
  E.g. :

  සමහරවිට vs සමහර විට
  කියනවනම් vs කියනව නම්
  කරනවාද vs කරනවා ද
  කළාය vs කළා ය

These type of errors massively contribute to the WER even if the transcriptions are good and readable. Hence, one can refer to the WER and mistakenly determine the usability of the ASR system as poor while the actual transcription seem to be acceptable.

E.g. :
Reference : ඔහු කණස්සල්ලට පත් වූයේ පුංචිමැණිකා සිහි වීමෙනි
Prediction : ඔහු කනස්සල්ලට පත්වූයේ පුංචි මැණිකා සිහිවීමෙනි
WER : 85%

Reference : මේ ලියුම් පත් බොහෝම කාලයක සිට පාවිච්චි කරනවා ද
Prediction : මේ ලියුම්පත් බොහෝම කාලයක සිට පාවිච්චි කරනවාද
WER : 44%

Reference : මම දිවි නසාගෙන නුඹ මේ සියල්ලෙන් නිදහස් කරන්නම්
Prediction : මම දිවි නසාගෙන නුඹමේ සියල්ලෙන් නිදහස් කරන්නම්
WER : 25%

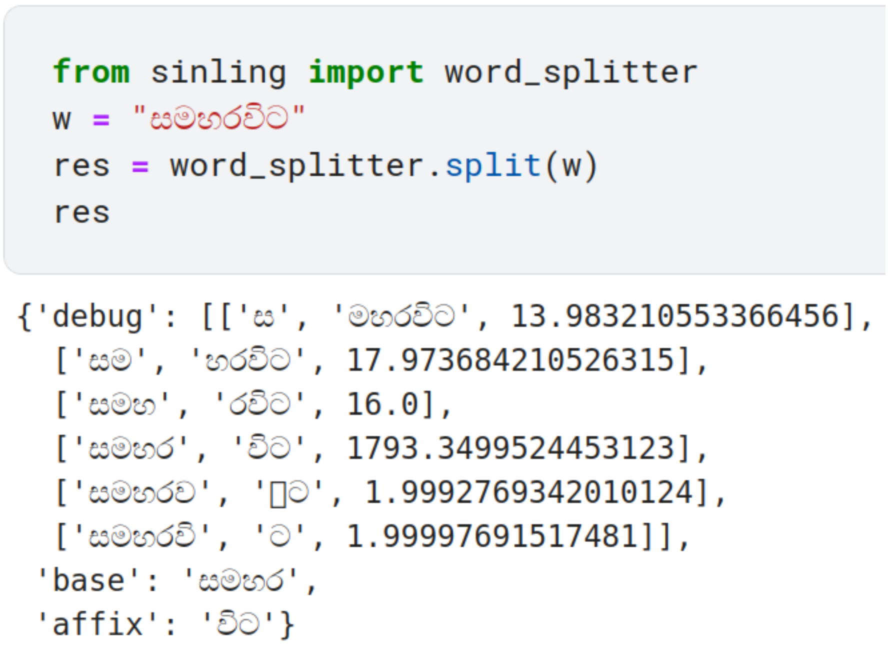
The above example shows how the WER of the sentence is calculated to be 85% due to spelling and word boundary errors while the transcription is quite acceptable.

In order to mitigate the kinds of errors mentioned above, a stage of post-processing has been introduced. Many researchers have used post-processing of the transcriptions received from an ASR system to correct the similar errors mentioned above and increase the readability of the transcriptions and proved the method to be effective (Liao et al. 2023; Bassil and Alwani 2012).

In this research, to correct the spelling errors and the word boundary errors, the previously created LM is used.

For spelling errors, each word in the transcriptions is scored with the language model to determine if the word is in the LM or not. If a word does not appear in the LM, a set of words which are closer to the wrong word will be taken from the set of uni-grams (single words) of the LM. To check the closeness, Levenshtein distance is calculated. Then each word in the set of close words is replaced with the wrong word and a set of possible transcriptions is created. Then each transcription in the set is again scored using the LM and the transcription with the best score is obtained. Even if this method is not as sophisticated as a rule-based or neural network based spell checker, it still resolved an acceptable amount of errors in the transcriptions.

In order to correct the word boundary issues, the "sinling" python library which contains several language processing tools for Sinhala is used. This library provides a method called "word_splitter" which morphologically splits the words into two words. An example is shown in the figure 4.3.



```
from sinling import word_splitter
w = "සමහරවිට"
res = word_splitter.split(w)
res
```

```
{'debug': [['ස', 'මහරවිට', 13.983210553366456],
  ['සම', 'හරවිට', 17.973684210526315],
  ['සමහ', 'රවිට', 16.0],
  ['සමහර', 'විට', 1793.3499524453123],
  ['සමහරව', '□ට', 1.9992769342010124],
  ['සමහරවි', 'ට', 1.99997691517481]],
 'base': 'සමහර',
 'affix': 'විට'}
```

Figure 4.3: An example result of the "word_splitter" function

Each word in the transcription is split into two words using the above function.

Those two words are scored using the LM to see whether they are meaningful words (exists in the LM). If the two words are meaningful, a new sentence is created with the split words replacing the original word. This new transcription is then scored against the original transcription to determine the best transcription.

4.2.8 Data Augmentation

Data augmentation in Automatic Speech Recognition (ASR) refers to techniques used to artificially expand the diversity of training data by introducing variations in the speech signals without changing their underlying meaning. This is crucial for improving the robustness and generalization capabilities of ASR systems, especially when dealing with limited training data or to mitigate overfitting. Since, Nanayakkara and Weerasinghe (2023) has successfully used data augmentation to optimize their mono-lingual transfer learning model, it is beneficial to test it out for the multi-lingual models as well.

The same settings used by Nanayakkara and Weerasinghe (2023) for the applied augmentations has been used in this research since they were proven to be effective. Mozilla's DeepSpeech provides a tool to create an augmented dataset. However, this tool only supports overlay, codec, reverb, resample and volume augmentations. Hence, only these augmentations are applied to the dataset. The following command is used to create the augmented dataset using DeepSpeech.

```
bin/data_set_tool.py \
--augment overlay[p=0.3,source=/hdd/2019CS125/final_dataset/LSD/
   train/noise.csv,layers=10:1,snr=50:20~9] \
--augment reverb[p=0.1,delay=50.0~30.0,decay=10.0:2.0~1.0] \
--augment codec[p=0.4,bitrate=48000:16000] \
--augment volume[p=0.4,dbfs=-10:-40] \
test.csv test-augmented.tar.gz
```

Chapter 5

Results and Analysis

This chapter will discuss the results of the experimentation done in this research. First section of the chapter will introduce the evaluation metrics used throughout the research and the second section will discuss the results obtained from the experimentation and the analysis on them.

5.1 Evaluation Metrics

Evaluation metrics serve a key role in analyzing the performance of various systems, notably in disciplines like natural language processing and speech recognition. Among the diversity of metrics used, two typically employed ones are Word Error Rate (WER) and Character Error Rate (CER).

5.1.1 Word Error Rate (WER)

Word Error Rate (WER) is a statistic used to evaluate the accuracy of ASR systems. It evaluates the percentage of words that are improperly recognized or transcribed compared to a reference transcript.

The Word Error Rate (WER) calculation formula is given by:

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

Where:

- S is the number of word substitutions (words in the recognized transcript that are different from the reference transcript).
- D is the number of word deletions (words missing in the recognized transcript compared to the reference transcript).
- I is the number of word insertions (extra words in the recognized transcript compared to the reference transcript).
- N is the total number of words in the reference transcript.

After calculating the WER using this formula, the result is typically multiplied by 100 to express it as a percentage.

5.1.2 Character Error Rate (CER)

Character Error Rate (CER) is another key parameter used in evaluating the performance of ASR systems. Unlike WER, which focuses on words, CER evaluates the percentage of characters that are erroneously recognized or transcribed compared to a reference transcript. Similar to WER, CER considers insertions, deletions, and replacements but acts at the character level.

The Character Error Rate (CER) calculation formula is given by:

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

Where:

- S is the number of character substitutions (characters in the recognized transcript that are different from the reference transcript).
- D is the number of character deletions (characters missing in the recognized transcript compared to the reference transcript).
- I is the number of character insertions (extra characters in the recognized transcript compared to the reference transcript).
- N is the total number of characters in the reference transcript.

After calculating the CER using this formula, the result is typically multiplied by 100 to express it as a percentage.

5.2 Experiments, Results and Analysis

5.2.1 Datasets

For the purpose of evaluating the models created in this research, two different datasets have been used. One of the two datasets utilized is the dataset created by Gamage, Pushpananda, Weerasinghe, et al. (2020) which included recordings from 4 female speakers and 4 male speakers where they utter 80 speech sentences altogether. The other dataset is a Sinhala only subset of the dataset created by Kjartansson et al. (2018) which is a crowd-sourced speech dataset for Javanese, Sundanese, Sinhala, Nepali, and Bangladeshi Bengali. The two datasets are utilized as follows.

- Dataset 1: The whole dataset by Gamage, Pushpananda, Weerasinghe, et al. (2020) with 80 utterances
- Dataset 2: A subset of 100 utterances from the dataset by Kjartansson et al. (2018)

From here onwards, the two datasets will be referred to as Dataset 1 and Dataset 2.

5.2.2 Results of the End-to-End Baseline Model

As mentioned earlier, the end-to-end model created by Gamage, Pushpananda, Nadungodage, et al. (2021) will be recreated in this research as a baseline model. However, rather than recreating the exact model developed by Gamage, Pushpananda, Nadungodage, et al. (2021), the version which was developed by Nanayakkara and Weerasinghe (2023) as a baseline model for their research will be implemented as they have managed to get a slightly better accuracy using plateau handling. Mozilla's DeepSpeech engine has been used to develop the end-to-end model. Table 5.1 shows the obtained results from the end-to-end baseline model by testing on the two datasets.

Dataset	WER(%)	CER(%)
Dataset 1	35.00	8.27
Dataset 2	41.62	16.34

Table 5.1: Results obtained by the end-to-end baseline model

5.2.3 Results of the Mono-lingual Baseline Model

As discussed in the section 4.1.2, Mozilla's DeepSpeech engine has been used to develop the mono-lingual model. A pre-trained DeepSpeech model for English language which has been trained using the Librispeech dataset (Panayotov et al. 2015), has been used as a source model similar to the original research. Data augmentation steps followed by Nanayakkara and Weerasinghe (2023) in their best performing model, which resulted in 17.19% WER, has been followed in the exact way when recreating this baseline model. Table 5.2 shows the obtained results from the mono-lingual baseline model by testing on the two datasets.

Dataset	WER(%)	CER(%)
Dataset 1	18.55	6.77
Dataset 2	40.48	15.54

Table 5.2: Results obtained by the mono-lingual baseline model

It is clear that even if this model performed well on dataset 1, It has higher WERs on the dataset 2. The dataset 2 having too many unseen speech data can be the reason for this difference in WER.

5.2.4 Results of the Multi-lingual Model

As discussed in the section 4.2, two models has been trained using the XLS-R pre-trained model as the source model. One model is trained using the 40 hour sinhala dataset which is available in the LTRL at UCSC. The other model uses data augmentation to apply some augmentations to the 40 hour dataset. Each model is evaluated under 3 conditions.

- The basic model
- The basic model + LM integrated
- The basic model + LM + Post-processing

The table 5.3 shows the results obtained by evaluating the multi-lingual transfer learning model without data augmentation.

Condition	WER(%)	CER(%)
Basic Model	47.05	9.17
Basic Model + LM	33.78	6.95
Basic Model + LM + Post-processing	24.28	8.89

Table 5.3: Results obtained by the multi-lingual model for dataset 1

By integrating an LM to decode the transcriptions, 28% WER reduction has been achieved. Further, using the post-processing mechanisms explained in the section 4.2.7, another 28% WER reduction has been achieved. When examining the results in the table 5.3, an increase in CER can be observed when implementing post-processing. That might be due to wrongfully adding and deleting certain characters to change the spellings.

A similar behaviour can be observed from the table 5.4, which shows the results obtained from testing the multi-lingual with the dataset 2.

Condition	WER(%)	CER(%)
Basic Model	55	12.69
Basic Model + LM	38.31	10.32
Basic Model + LM + Post-processing	36.6	12.16

Table 5.4: Results obtained by the multi-lingual model for dataset 2

Table 5.5 shows the results obtained from the multi-lingual model which was trained with augmented data and tested on the dataset 1.

Condition	WER(%)	CER(%)
Basic Model	48.56	10.18
Basic Model + LM	37.7	7.88
Basic Model + LM + Post-processing	26.54	7.58

Table 5.5: Results obtained by the multi-lingual model with data augmentation for dataset 1

Table 5.6 shows the results obtained from the multi-lingual model which was trained with augmented data and tested on the dataset 2.

Condition	WER(%)	CER(%)
Basic Model	54.58	12.68
Basic Model + LM	38.82	9.88
Basic Model + LM + Post-processing	35.29	10.94

Table 5.6: Results obtained by the multi-lingual model with data augmentation for dataset 2

Even if a slight increase in WER can be witnessed compared to the model without data augmentation, the effect from extensive post-processing, which was notable in Tables 5.3 and 5.4 doesn't seem to occur here as the CER has gone down in each condition. Less amount of substitutions in the transcriptions of the data augmentation based model might be the reason for this. Less amount of substitutions means there are less amount of spelling errors, thus post-processing will not have to correct them and increase the CER in the process.

5.2.5 Analysis

LM integration and post-processing

Analysing the above results, it is evident that integrating an LM and conducting post-processing does in deed contribute towards decreasing the WER. However, the effectiveness of those methods can be truly examined by looking at the transcriptions.

Condition	Transcript
Utterance	ඔහු කණස්සල්ලට පත් වූයේ පුංචිමැණිකා සිහි වීමෙනි
Basic Model	ඔහු කනස්සලලට පත් වූයේ කුංචි මැනිකා සිහිවීමෙනි
Basic Model + LM	ඔහු කනස්සලට පත් වූයේ කුංචි මැණිකා සිහිවීමෙනි
Basic Model + LM + PP	ඔහු කනස්සලට පත් වූයේ පුංචිමැණිකා සිහි වීමෙනි

Table 5.7: Results from LM and post-processing - sentence 1

Analysing the table 5.7, it can be noticed that, the word separation issues have been successfully handled by post-processing. The word "කුංචි" is also corrected as "පුංචි" due to the post-processing phase. The LM has managed to identify the error in "මැනිකා" and successfully corrected it to "මැණිකා".

By observing the results in section 5.2.4, it was notable that post-processing some times lead into increasing the CER. Table 5.8 shows a similar instance.

Condition	Transcript
Utterance	පීටර් ගැන කියනවා නම් මට එයාව අත් අරින්න ඕනෙ නෑ
Basic Model	පීටර් ගැන කියෙනවානම් මට යාව අත්හයින්න ඕනෙ නෑ
Basic Model + LM	පීටර් ගැන කියෙනවානම් මට එයාව අත්හයින්න ඕනෙ නෑ
Basic Model + LM + PP	පීටර් ගැන ගිල්ලන්න මට එයාව බලන්න ඕනෙ නෑ

Table 5.8: Results from LM and post-processing - sentence 2

It can be observed that, the word "කියෙනවානම්" is changed into "ගිල්ලන්න". And the word "අත්හයින්න" is changed into "බලන්න". These kind of errors happen due to overdoing the pre-processing phase. Iterating the spell corrector multiple times may change the incorrectly spelled words into something completely different. This might not affect the WER since the replaced word is wrong in the first place. However, this can greatly affect the CER since the replaced word contains some correct characters which will be replaced by entirely different characters.

Comparison between models

Table 5.9 shows an instance where the multi-lingual models outperform the baseline model. The word "වික්‍රමනායක" is not identified correctly in the baseline model while the multi-lingual models successfully identify it.

Model	Transcript
Utterance	වික්‍රමනායක මහත්තයාට කිවුවාම ඕනෑ දෙයක් කර දේවි
Baseline	වක් රවා මහත්තයට කිව්වාම ඕන දයක් කරේ වි
Multi-lingual	වික්‍රමනායක මහත්තයට කිව්වාම ඕන දෙයක් කර දේවි
Multi-lingual + DA	වික්‍රමනායක මහත්තයට කිව්වා ම ඕන දෙයක් කර දේවි

Table 5.9: Comparison of results between models - sentence 1

Words that sounds quite similar, are a real challenge for ASR models to decode. For an example, "බැළලිය" vs. "බැල්ලිය". Table 5.10 shows how each model identified the word "බැළලිය". It should be noted that both the models which identified the word correctly has incorporated data augmentation. That observation seems plausible since data augmentation increases the ability of ASR models to distinguish between subtle differences in speech.

Model	Transcript
Utterance	බැළලිය අසුනෙන් බිමට පැන දොරටුව දෙසට ගමන් කළා ය
Baseline	බැළලිය අසුනෙන් බිමට පැන දොරටුව දෙසට ගමන් කළා ය
Multi-lingual	බැල්ලිය අසුනෙන් බිමට පැන දොරටුව දෙසට ගමන් කළා ය
Multi-lingual + DA	බැළලිය අසුනෙන් බිමට පැන දොරටුව දෙසට ගමන් කළා ය

Table 5.10: Comparison of results between models - sentence 2

Table 5.11 shows an example where multi-lingual model with data augmentations applied outperformed both the baseline and the multi-lingual model without data augmentations.

Model	Transcript
Utterance	මරතීනු උදැල්ල පසෙක තබා තිරික්කලය වෙතට පා එසවීය
Baseline	මරතීනු ගැන පසෙක තවත් තිරික්කලය වෙතට පා සවි
Multi-lingual	සැදැහැවත්හු දැල්ල පසෙක තබා තිරික්කලය වෙතට පා එසවීය
Multi-lingual + DA	මරතීනු උදැල්ල පසෙක තබා තිරික්කලය වෙතට පා එසවීය

Table 5.11: Comparison of results between models - sentence 3

The final transcription achieved by the multi-lingual model with data augmentation on the sentence "මරතීනු උදැල්ල පසෙක තබා තිරික්කලය වෙතට පා එසවීය" was achieved with the help of the LM and the post-processing. Table 5.12 shows the evolution of the sentence through the LM and the post-processing phase.

Condition	Transcript
Utterance	මරතීනු උදැල්ල පසෙක තබා තිරික්කලය වෙතට පා එසවීය
Basic Model	මරතීන් ඉදැල්ල පසෙක තබත් සිරික් කළේය වෙතර පා එස වීය
Basic Model + LM	මරතීන් ඉදැල්ල පසෙක තබත් සිරික්කළේය වෙතට පා එසවීය
Basic Model + LM + PP	මරතීනු උදැල්ල පසෙක තබා තිරික්කලය වෙතට පා එසවීය

Table 5.12: Results from LM and post-processing - sentence 3

Chapter 6

Conclusion

This dissertation is about testing the applicability of transfer learning on end-to-end Sinhala speech recognition. Chapter 1 contains a brief introduction to automatic speech recognition (ASR), the technologies that have been used throughout the history and the current state of Sinhala ASR. In chapter 2, a comprehensive literature review has been conducted to outline the evolution of ASR and the gap between Sinhala ASR and other ASR systems in terms of the technologies used. Chapter 3 contains in-depth information on the technologies, architectures and algorithms used in this research. In chapter 4, the step by step approach taken in this research has been outlined. Chapter 5 displays the results from the experimentation done throughout the research and an analysis of the results. Finally, this chapter provides the conclusions drawn from the research.

6.1 Effectiveness of WER as an Evaluation Metric for Sinhala

When evaluating the performance of Automatic Speech Recognition (ASR) systems, metrics like Word Error Rate (WER) and Character Error Rate (CER) are commonly used. These metrics help gauge how accurately spoken words are transcribed into text.

WER is often considered a primary measure of ASR system quality. A lower WER is generally interpreted as indicating better performance in converting speech to text. However, for Sinhala, relying solely on WER for evaluation may not reflect how well the ASR system performs.

Sinhala has unique linguistic features that may not align well with how WER works. For example, spoken words may be combined but written separately, leading to discrepancies between spoken and written forms and inflating WER scores.

Set	Prediction and Reference
1	Reference: මේ ලියුම් පත් බොහෝම කාලයක සිට පාවිච්චි කරනවා ද
	Prediction: මේ ලියුම්පත් බොහෝම කාලයක සිට පාවිච්චි කරනවාද
2	Reference: මසුරු සිටානන්ගේ පුතා නන්ද සිදුහත් දන්නවා නෙව ද
	Prediction: මසුරු සිටානන්ගේ පුතා නන්ද සිදුහත් දන්නවා නෙවද
3	Reference: තමාගේ සිතීමේ ශක්තිය නතර වී ඇත්තා සේ ඔහුට හැඟිණ
	Prediction: තමාගේ සිතීමේ ශක්තිය නතරවී ඇත්තාසේ ඔහුට හැඟින

Table 6.1: Example word boundary differences when spoken and written

Table 6.1 contains a set of example predictions and references. In set 1, the word “ලියුම් පත්” is the reference, while the prediction gave it as “ලියුම්පත්”. Also the word “කරනවා ද” has been predicted as “කරනවාද”. However, when considering the actual way a person who speaks Sinhala will pronounce those words, one will rarely pause between “කරනවා” and “ද” when speaking in a regular context. Hence, it is acceptable that the ASR system detect it as a single word despite the fact that the reference has it as two separate words. In sets 2 and 3, similar behaviour can be observed. Words “නෙවද”, “ඇත්තාසේ”, and “නතරවී” are often pronounced as a single word without pausing in between. Although the ASR system results in a higher WER due to these errors, when the system is used in a practical context, without having a written version of the spoken text, the resultant predictions are of an acceptable quality in terms of the accuracy.

To address these challenges, researchers must look beyond WER when evaluating Sinhala ASR systems. Alternative measures, such as analyzing word structure and assessing coherence between spoken words and their intended meaning, can provide a more comprehensive evaluation. Utilizing specific criteria and datasets tailored for Sinhala can also improve the assessment of ASR system performance. Additionally, as this research has proven, post-processing the ASR output can help evaluating the usability of an ASR system.

Ultimately, while WER remains a useful metric, it should not be relied upon exclusively for evaluating ASR systems for Sinhala. By acknowledging these complexities and employing a diverse range of evaluation methods, researchers can develop more effective ASR systems that better serve the needs of Sinhala speakers.

6.2 Conclusion about Research Questions

There were two research questions formed in this research.

- How can multi-lingual transfer learning models be used to improve Sinhala ASR?
- How to optimize a Sinhala ASR model to achieve the best accuracy while being resource efficient?

In chapter 4, the step by step approach for fine-tuning the XLS-R model on Sinhala data has been outlined. Chapter 5 contains the analysis of the results of the experimentation conducted throughout the research. By observing the results, it can be seen that the multi-lingual model displayed an acceptable results where in some evaluations it even outperformed the baseline models in terms of the WER. Apart from the WER, by looking at some of the actual transcriptions which are presented in section 5.2.4, the multi-lingual ASR model presented in this study can be seen as of acceptable quality in terms of the accuracy of the transcriptions. Hence, it is safe to conclude the research question “How can multi-lingual transfer learning models be used to improve Sinhala ASR?”.

The research question, “How to optimize a Sinhala ASR model to achieve the best accuracy while being resource efficient?” refers to finding out any limitations which might arise after developing a multi-lingual transfer learning model for Sinhala language and finding out how to solve them and by doing so optimizing the developed model while being resource efficient. As mentioned earlier in this study, two kinds of errors were seemed to be occurring in the output transcriptions of the multi-lingual model. The spelling errors and the word boundary errors. Those were the main limitations identified in the initial model. Through experimentation with integrating language models and conducting a phase of post-processing to correct the spelling and word boundary errors, it is safe to conclude this research question as well.

6.3 Limitations

There were few limitations identified in the proposed multi-lingual model. When applying data augmentation, the types of augmentations were limited to overlay, codec, reverb, resample and volume augmentations since DeepSpeech was used to apply the augmentations and when creating a separate augmented dataset, DeepSpeech only allowed those augmentations. Other augmentations such as pitch, tempo, frequency mask, time mask, dropout, and etc. was not experimented. Additionally, the recognition of named-entities were poor compared to other words in the proposed model.

6.4 Future Directions

In this research, two kinds of post-processing techniques are tested and were proven to be effective. There are other possible post-processing techniques such as grammar correction and spell correction using more sophisticated algorithms.

Another room for improvement is improving the accuracy of named-entity recognition. Improving the speech corpus with more named-entities or having a separate text corpus with named-entities and correcting the errors in the post-processing phase might be starting points for a solution.

Since Sinhala speakers often speak in mixed tongues (Sinhala mixed with English), ASR systems which provides transcriptions in two languages can be explored. XLS-R would be of help since it has been trained using multiple languages.

In recent studies, Meta Learning has been used for fast adaptations for transfer learning ASR models while keeping the model complexity significantly minimum (Hou et al. 2021). It would be beneficial to study about that.

References

Arisaputra, Panji et al. (2024). “XLS-R Deep Learning Model for Multilingual ASR on Low-Resource Languages: Indonesian, Javanese, and Sundanese”. In: *arXiv preprint arXiv:2401.06832*.

Arora, Shipra J and Rishi Pal Singh (2012). “Automatic speech recognition: a review”. In: *International Journal of Computer Applications* 60.9.

Babu, Arun et al. (2021). “XLS-R: Self-supervised cross-lingual speech representation learning at scale”. In: *arXiv preprint arXiv:2111.09296*.

Baevski, Alexei et al. (2020). “wav2vec 2.0: A framework for self-supervised learning of speech representations”. In: *Advances in neural information processing systems* 33, pp. 12449–12460.

Bassil, Youssef and Mohammad Alwani (2012). “Post-editing error correction algorithm for speech recognition using bing spelling suggestion”. In: *arXiv preprint arXiv:1203.5255*.

Boersma, Paul and Vincent Van Heuven (2001). “Speak and unSpeak with PRAAT”. In: *Glot International* 5.9/10, pp. 341–347.

Bozinovski, Stevo and Ante Fulgosi (1976). “The influence of pattern similarity and transfer learning upon training of a base perceptron b2”. In: *Proceedings of Symposium Informatica*. Vol. 3, pp. 121–126.

Cho, Jaejin et al. (2018). “Multilingual sequence-to-sequence speech recognition: architecture, transfer learning, and language modeling”. In: *2018 IEEE Spoken Language Technology Workshop (SLT)*. IEEE, pp. 521–527.

Conneau, Alexis et al. (2020). “Unsupervised cross-lingual representation learning for speech recognition”. In: *arXiv preprint arXiv:2006.13979*.

Dahl, George E et al. (2011). “Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition”. In: *IEEE Transactions on audio, speech, and language processing* 20.1, pp. 30–42.

Dalmia, Siddharth et al. (2018). “Sequence-based multi-lingual low resource speech recognition”. In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, pp. 4909–4913.

Davis, Ken H et al. (1952). “Automatic recognition of spoken digits”. In: *The Journal of the Acoustical Society of America* 24.6, pp. 637–642.

Denes, Pablo (1959). “The design and operation of the mechanical speech recognizer at University College London”. In: *Journal of the British Institution of Radio Engineers* 19.4, pp. 219–229.

Devlin, Jacob et al. (2018). “Bert: Pre-training of deep bidirectional transformers for language understanding”. In: *arXiv preprint arXiv:1810.04805*.

Forgie, James W and Carma D Forgie (1959). “Results obtained from a vowel recognition computer program”. In: *The Journal of the Acoustical Society of America* 31.11, pp. 1480–1489.

Fry, Dennis Butler (1959). “Theoretical aspects of mechanical speech recognition”. In: *Journal of the British Institution of Radio Engineers* 19.4, pp. 211–218.

Gamage, Buddhi, Randil Pushpananda, Thilini Nadungodage, et al. (2021). “Improve Sinhala Speech Recognition Through e2e LF-MMI Model”. In: *Proceed-*

ings of the 18th International Conference on Natural Language Processing (ICON), pp. 213–219.

Gamage, Buddhi, Randil Pushpananda, Ruvan Weerasinghe, et al. (2020). "Usage of combinational acoustic models (dnn-hmm and sgmm) and identifying the impact of language models in sinhala speech recognition". In: *2020 20th International Conference on Advances in ICT for Emerging Regions (ICTer)*. IEEE, pp. 17–22.

Graves, Alex et al. (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.

Haldar, Rishin and Debajyoti Mukhopadhyay (2011). "Levenshtein distance technique in dictionary lookup methods: An improved approach". In: *arXiv preprint arXiv:1101.1232*.

Heafield, Kenneth (2011). "KenLM: Faster and smaller language model queries". In: *Proceedings of the sixth workshop on statistical machine translation*, pp. 187–197.

Hou, Wenxin et al. (2021). "Meta-adapter: Efficient cross-lingual adaptation with meta-learning". In: *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, pp. 7028–7032.

Huang, Jui-Ting et al. (2013). "Cross-language knowledge transfer using multilingual deep neural network with shared hidden layers". In: *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE, pp. 7304–7308.

Jang, Eric et al. (2016). "Categorical reparameterization with gumbel-softmax". In: *arXiv preprint arXiv:1611.01144*.

Jegou, Herve et al. (2010). "Product quantization for nearest neighbor search". In: *IEEE transactions on pattern analysis and machine intelligence* 33.1, pp. 117–128.

Juang, Biing Hwang and Laurence R Rabiner (1991). "Hidden Markov models for speech recognition". In: *Technometrics* 33.3, pp. 251–272.

Karunanayake, Yohan et al. (2019). "Transfer learning based free-form speech command classification for low-resource languages". In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: Student Research Workshop*, pp. 288–294.

Karunathilaka, Hirunika et al. (2020). "Low-resource Sinhala speech recognition using deep learning". In: *2020 20th International Conference on Advances in ICT for Emerging Regions (ICTer)*. IEEE, pp. 196–201.

Kjartansson, Oddur et al. (Aug. 2018). "Crowd-Sourced Speech Corpora for Javanese, Sundanese, Sinhala, Nepali, and Bangladeshi Bengali". In: *Proc. The 6th Intl. Workshop on Spoken Language Technologies for Under-Resourced Languages (SLTU)*. Gurugram, India, pp. 52–55. URL: `http://dx.doi.org/10.21437/SLTU.2018-11`.

Krishna, DN et al. (2021). "Using Large Self-Supervised Models for Low-Resource Speech Recognition." In: *Interspeech*, pp. 2436–2440.

Lee, Kai-Fu (1988). “On large-vocabulary speaker-independent continuous speech recognition”. In: *Speech communication* 7.4, pp. 375–379.

Li, Jinyu et al. (2022). “Recent advances in end-to-end automatic speech recognition”. In: *APSIPA Transactions on Signal and Information Processing* 11.1.

Liao, Junwei et al. (2023). “Improving readability for automatic speech recognition transcription”. In: *ACM Transactions on Asian and Low-Resource Language Information Processing* 22.5, pp. 1–23.

Nadungodage, Thilini and Ruvan Weerasinghe (2011). “Continuous sinhala speech recognizer”. In: *Conference on Human Language Technology for Development, Alexandria, Egypt.* Citeseer, pp. 2–5.

Nagata, Kuniichi et al. (1964). “Spoken digit recognizer for the Japanese language”. In: *Journal of the Audio Engineering Society* 12.4, pp. 336–342.

Nanayakkara, Lakshika and Ruvan Weerasinghe (2023). “Exploring Model-Level Transfer Learning to Improve the Recognition of Sinhala Speech”. In: *International Conference on Machine Learning, Deep Learning and Computational Intelligence for Wireless Communication.* Springer, pp. 17–28.

Olson, Harry F and Herbert Belar (1956). “Phonetic typewriter”. In: *The Journal of the Acoustical Society of America* 28.6, pp. 1072–1081.

Pammi, Sathish et al. (2010). “Multilingual Voice Creation Toolkit for the MARY TTS Platform.” In: *LREC.* Citeseer.

Pan, Sinno Jialin and Qiang Yang (2009). “A survey on transfer learning”. In: *IEEE Transactions on knowledge and data engineering* 22.10, pp. 1345–1359.

Panayotov, Vassil et al. (2015). “Librispeech: An ASR corpus based on public domain audio books”. In: *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5206–5210. DOI: `10.1109/ICASSP.2015.7178964`.

Rabiner, L et al. (1979). “Speaker-independent recognition of isolated words using clustering techniques”. In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 27.4, pp. 336–349.

Sakai, Toshiyuki (1962). “The phonetic typewriter”. In: *Proc. IFIP Congress 62*, pp. 445–450.

Tan, Chuanqi et al. (2018). “A survey on deep transfer learning”. In: *Artificial Neural Networks and Machine Learning–ICANN 2018: 27th International Conference on Artificial Neural Networks, Rhodes, Greece, October 4-7, 2018, Proceedings, Part III 27.* Springer, pp. 270–279.

Vaswani, Ashish et al. (2017). “Attention is all you need”. In: *Advances in neural information processing systems* 30.

Wang, Dong et al. (2019). “An overview of end-to-end automatic speech recognition”. In: *Symmetry* 11.8, p. 1018.

Yi, Jiangyan et al. (2018). “Language-adversarial transfer learning for low-resource speech recognition”. In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 27.3, pp. 621–630.

Yu, Chongchong et al. (2019). “Cross-language end-to-end speech recognition research based on transfer learning for the low-resource Tujia language”. In: *Symmetry* 11.2, p. 179.

Appendix A

Code Listings

Listing 1: Full training code

```
from datasets import load_dataset, load_from_disk, load_metric,
   Audio, ClassLabel, Value, Audio, Dataset, Features, DatasetDict
from IPython.display import display, HTML
from transformers import Wav2Vec2ForCTC, TrainingArguments, Trainer
from transformers import Wav2Vec2CTCTokenizer,
   Wav2Vec2FeatureExtractor, Wav2Vec2Processor, Wav2Vec2ForCTC
from dataclasses import dataclass, field
from typing import Any, Dict, List, Optional, Union
from huggingface_hub import create_repo, notebook_login
import random
import pandas as pd
import re
import json
import os
import torch
import numpy as np
import librosa

# Define features
features = Features(
    {
        "path": Value("string"),
        "size": Value("int32"),
        "sentence": Value("string"),
        "audio": Audio(sampling_rate=16000),
    })

# Load Sinhala dataset
sinhala_data = load_dataset(
    'csv', data_files={
        'train': '/kaggle/input/sinhala-asr-data/train_kaggle1_full.
            csv',
        'test': '/kaggle/input/sinhala-asr-data/test_kaggle_new.csv'
    }
)

# Cast dataset features
sinhala_data = sinhala_data.cast(features)

# Remove unnecessary columns
```

```python
sinhala_data["train"] = sinhala_data["train"].remove_columns(["size"])
sinhala_data["test"] = sinhala_data["test"].remove_columns(["size"])

# Define regex pattern for special characters removal
chars_to_remove_regex = '[\)…"‘" \(\\,\?\.\!\-\;\:\"\\%\\\\']'

# Remove special characters function
def remove_special_characters(batch):
    batch["sentence"] = re.sub(chars_to_remove_regex, '', batch["sentence"])
    return batch

# Apply special characters removal to train and test datasets
sinhala_data["train"] = sinhala_data["train"].map(remove_special_characters)
sinhala_data["test"] = sinhala_data["test"].map(remove_special_characters)

# Extract vocabulary
def extract_all_chars(batch):
  all_text = "␣".join(batch["sentence"])
  vocab = list(set(all_text))
  return {"vocab": [vocab], "all_text": [all_text]}

vocab_train = sinhala_data["train"].map(extract_all_chars, batched=True, batch_size=-1, keep_in_memory=True, remove_columns=sinhala_data["train"].column_names)
vocab_test = sinhala_data["test"].map(extract_all_chars, batched=True, batch_size=-1, keep_in_memory=True, remove_columns=sinhala_data["test"].column_names)
vocab_list = list(set(vocab_train["vocab"][0]) | set(vocab_test["vocab"][0]))
vocab_dict = {v: k for k, v in enumerate(sorted(vocab_list))}

vocab_dict["|"] = vocab_dict["␣"]
del vocab_dict["␣"]

vocab_dict["[UNK]"] = len(vocab_dict)
vocab_dict["[PAD]"] = len(vocab_dict)

# Save vocabulary to a JSON file
with open('vocab.json', 'w') as vocab_file:
    json.dump(vocab_dict, vocab_file, ensure_ascii=False)
```

```python

# Initialize tokenizer and push to Hugging Face Hub
tokenizer = Wav2Vec2CTCTokenizer.from_pretrained("/kaggle/working/", unk_token="[UNK]", pad_token="[PAD]", word_delimiter_token="|")
repo_name = "wav2vec2-large-xls-r-300m-sinhala-original-split-part4-epoch30-final"
tokenizer.push_to_hub(repo_name)

# Initialize feature extractor
feature_extractor = Wav2Vec2FeatureExtractor(feature_size=1, sampling_rate=16000, padding_value=0.0, do_normalize=True, return_attention_mask=True)

# Initialize processor
processor = Wav2Vec2Processor(feature_extractor=feature_extractor, tokenizer=tokenizer)

# Prepare dataset function
def prepare_dataset(batch):
    audio = batch["audio"]

    # Un-batch batched output
    batch["input_values"] = processor(audio["array"], sampling_rate=audio["sampling_rate"]).input_values[0]
    batch["input_length"] = len(batch["input_values"])

    with processor.as_target_processor():
        batch["labels"] = processor(batch["sentence"]).input_ids
    return batch

# Cache directories for train and test datasets
cache_dir_train = "common_voice_train_cache"
cache_dir_test = "common_voice_test_cache"

# Preprocess and save train dataset if not already cached
if not os.path.exists(cache_dir_train):
    print("common_voice_train_cache␣does␣not␣exist.␣Preprocessing␣and␣saving␣to␣disk.")
    sinhala_data["train"] = sinhala_data["train"].map(prepare_dataset, remove_columns=sinhala_data["train"].column_names)
    sinhala_data["train"].save_to_disk(cache_dir_train)
else:
    print("common_voice_train_cache␣exists.␣Loading␣from␣disk.")
    sinhala_data["train"] = load_from_disk(cache_dir_train)

# Preprocess and save test dataset if not already cached
if not os.path.exists(cache_dir_test):
    print("common_voice_test_cache␣does␣not␣exist.␣Preprocessing␣and␣saving␣to␣disk.")
    sinhala_data["test"] = sinhala_data["test"].map(prepare_dataset, remove_columns=sinhala_data["test"].column_names)
    sinhala_data["test"].save_to_disk(cache_dir_test)
else:
    print("common_voice_test_cache␣exists.␣Loading␣from␣disk.")
    sinhala_data["test"] = load_from_disk(cache_dir_test)

# Data collator for CTC with padding
@dataclass
class DataCollatorCTCWithPadding:
    processor: Wav2Vec2Processor
    padding: Union[bool, str] = True

    def __call__(self, features: List[Dict[str, Union[List[int], torch.Tensor]]]) -> Dict[str, torch.Tensor]:
        input_features = [{"input_values": feature["input_values"]} for feature in features]
        label_features = [{"input_ids": feature["labels"]} for feature in features]

        batch = self.processor.pad(
            input_features,
            padding=self.padding,
            return_tensors="pt",
        )
        with self.processor.as_target_processor():
            labels_batch = self.processor.pad(
                label_features,
                padding=self.padding,
                return_tensors="pt",
            )

        labels = labels_batch["input_ids"].masked_fill(labels_batch.attention_mask.ne(1), -100)

        batch["labels"] = labels

        return batch

```

```python
# Initialize data collator
data_collator = DataCollatorCTCWithPadding(processor=processor,
    padding=True)

# Load WER metric
wer_metric = load_metric("wer")

# Load pre-trained model
model = Wav2Vec2ForCTC.from_pretrained(
    "facebook/wav2vec2-xls-r-300m",
    attention_dropout=0.0,
    hidden_dropout=0.0,
    feat_proj_dropout=0.0,
    mask_time_prob=0.05,
    layerdrop=0.0,
    ctc_loss_reduction="mean",
    pad_token_id=processor.tokenizer.pad_token_id,
    vocab_size=len(processor.tokenizer),
)

# Compute metrics function
def compute_metrics(pred):
    pred_logits = pred.predictions
    pred_ids = np.argmax(pred_logits, axis=-1)

    pred.label_ids[pred.label_ids == -100] = processor.tokenizer.
        pad_token_id

    pred_str = processor.batch_decode(pred_ids)
    label_str = processor.batch_decode(pred.label_ids, group_tokens
        =False)

    wer = wer_metric.compute(predictions=pred_str, references=
        label_str)

    return {"wer": wer}

# Freeze feature extractor
model.freeze_feature_extractor()

# Training arguments
training_args = TrainingArguments(
  output_dir=repo_name,
  group_by_length=True,
  hub_strategy="checkpoint",
  per_device_train_batch_size=8,
  gradient_accumulation_steps=2,
  evaluation_strategy="steps",
  num_train_epochs=30,
  gradient_checkpointing=True,
  save_steps=400,
  eval_steps=400,
  logging_steps=400,
  learning_rate=3e-4,
  warmup_steps=500,
  save_total_limit=2,
  push_to_hub=True,
)

# Initialize trainer
trainer = Trainer(
    model=model,
    data_collator=data_collator,
    args=training_args,
    compute_metrics=compute_metrics,
    train_dataset=sinhala_data["train"],
    eval_dataset=sinhala_data["test"],
    tokenizer=processor.feature_extractor,
)

# Train the model
trainer.train()

# Push trained model to Hugging Face Hub
trainer.push_to_hub()
```