Translation of Afan Oromo Sign Language to Afan Oromo Text Using Deep Learning Approach

Abdi Damtew Bezu

A Thesis Submitted to the Department of Computer Science and Engineering, School of Electrical Engineering and Computing

Presented in Partial Fulfillment of the Requirement for the Degree of Master's in Computer Science and Engineering

Office of Graduate Studies

Adama Science and Technology University

February 2025

Adama, Ethiopia

Translation of Afan Oromo Sign Language to Afan Oromo Text Using Deep Learning Approach

Abdi Damtew Bezu

Advisor: Teklu Urgessa(PhD)

A Thesis Submitted to the Department of Computer Science and Engineering,
School of Electrical Engineering and Computing

Presented in Partial Fulfillment of the Requirement for the Degree of Master's in Computer Science and Engineering

Office of Graduate Studies

Adama Science and Technology University

February 2025

Adama, Ethiopia

DECLARATION

I hereby declare that this Master's Thesis entitled "**Translation of Afan Oromo Sign Language to Afan Oromo Text using Deep Learning Approach**" is my original work. That is, it has not been submitted for the award of any academic degree, diploma, or certificate in any other university. All sources of materials that are used for this thesis have been duly acknowledged through citation.

Abdi Damtew Bezu	_____	_____
Name of the student	Signature	Date

RECOMMENDATION OF ADVISOR

I, the advisor of this thesis, hereby certify that I have read the revised version of the thesis entitled "**Translation of Afan Oromo Sign Language to Afan Oromo Text Using Deep Learning Approach**" prepared under my guidance by Abdi Damtew Bezu, submitted in partial fulfillment of the requirements for the degree of Master's of Science in Computer Science and Engineering. Therefore, I recommend the submission of a revised version of the thesis to the department following the applicable procedures.

Teklu Urgessa(PhD)	_____	_____
Name of the student	Signature	Date

Approval Sheet

I the advisor of the thesis entitled "**Translation of Afan Oromo Sign Language to Afan Oromo Text Using Deep Learning Approach**" developed by **Abdi Damtew Bezu,** hereby certify that the recommendation and suggestions made by the board of examiners are appropriately incorporated into the final version of the thesis.

Teklu Urgessa (Ph.D.)	_____	_____
Major Advisor	Signature	Date

ACKNOWLEDGMENT

First and foremost, I would like to thank the almighty God who helped me and made everything possible. Next, I would like to thank my Advisor Dr. Teklu Urgesa(Ph.D) for his encouragement, guidance, and advice. Next, my deepest thanks go to all my family members especially to my grandmother Itenesh Bogale, my mom Yirgalem Mamo and my sister Derartu Kefalo for their continuous and endless support throughout my life. Next, I would like to express my eternal gratitude to my uncles Girma M, Muluneh M and Mesfin M who helped and supported me in every step of this study, and in shaping my life.

Lastly, I would like to thank Adama Science and Technology University for giving me a chance to attend my post-graduate study for the past two years.

Table of Contents

LIST OF TABLES

LIST OF FIGURES

LIST OF ACRONYMS

AI	Artificial Intelligence
ANN	Artificial Neural Network
Bi-LSTM	Bi-directional Long Short Term Memory
BLEU	Bi-Lingual Evaluation Understudy
GAN	Generative Adverserial Network
GPU	Graphics Processing Unit
GRU	Gated Recurrent Unit
IDE	Integrated Development Environment
JPEG	Joint Photographic Experts Group
LSTM	Long Short Term Memory
mAP	Mean Average Precision
PNG	Portable Network Graphics
RNN	Recurrent Neural Network
ROUGE	Recall Oriented Understudy for Gisting Evaluation
SL	Sign Language
SLR	Sign Language Recognition
SLT	Sign Language Translation
SVM	Support Vector Machine
UML	Unified Modeling Language
WER	Word Error Rate
YOLOv5	You Only Look Once, Version 5

LIST OF SAMPLE CODES

Abstract

Sign language is a mode of communication used by individuals with hearing impairments and those with hearing difficulties to interact with one another and with individuals who can hear. However, Afan Oromo Sign Language has not been thoroughly studied, with no publicly available datasets and limited research efforts. Most existing studies on Afan Oromo Sign Language translation focus on sign recognition rather than complete sign-to-text translation. This research seeks to address this gap by developing an automatic translation system that converts Afan Oromo Sign Language into Afan Oromo text using LSTM and Transformer encoder-decoder models. A custom dataset was prepared by extracting image frames from videos of signed phrases, ensuring a signer-independent translation system. The study implemented two deep learning architectures: an LSTM-based sequence-to-sequence model and a Transformer-based encoder-decoder model. These models were evaluated using Word Error Rate (WER) and ROUGE (ROUGE-1, ROUGE-2, and ROUGE-L) scores. Results indicate that the LSTM model achieves lower WER (best: 18.34%), suggesting better sequence alignment, whereas the Transformer model excels in text generation quality, with ROUGE-1 reaching 58.95%. However, the higher WER of the Transformer model (42.17% - 48.49%) highlights challenges in structural consistency between sign sequences and textual outputs. This study marks a significant step forward in translating Afan Oromo Sign Language phrases and sentences and offers essential insights for developing low-resource sign language translation systems. The findings highlight the necessity for larger datasets, hybrid models, and linguistic adaptation techniques to improve translation accuracy and accessibility for the Deaf community.

Keywords: *Afan Oromo Sign Language, Sign Language Translation, LSTM, Transformer, Encoder-Decoder, Deep Learning, ROUGE, WER*

CHAPTER ONE

1. INTRODUCTION

1.1 Background of Study

Instead of spoken words, sign languages communicate through visual and manual expressions.They use hand movements and markers to deliver meaning. Similar to spoken languages, sign languages are complete and natural, possessing their own grammatical structures and vocabulary. Sign languages are great way of conveying thoughts and feelings for people with hearing trouble(Diriba,Etana & Hawi,2023). Sign language is primarily associated with deaf people and people of hard hearing, but it also serves purposes. It can be used as a communication tool for hearing individuals who are unable to speak, those with oral language difficulties, and people with family members, such as parents of deaf children.

It's difficult to give an exact figure for the number of sign languages globally. While most countries have their own unique sign language, some have multiple. Sources like Ethnologue (Eberhard et al, 2021) and the SIGN-HUB Atlas (Hoseman et al, 2021) provide estimates, but acknowledge that many sign languages are undocumented or undiscovered, suggesting the actual number of sign languages is higher. Stokoe , HamNoSys , SignWriting , and Gloss Notation are some of sign language forms (Mohamed, Hesahm & Ammar, 2021) . Stokoe notation is beneficial but lacks representation of non-manual components such as facial expressions and body movements, which limits its effectiveness for precise translation. HamNoSys, which is designed for 3D avatar animation, also fails to capture these non-manual components. SignWriting is visually descriptive but presents challenges for computer analysis. In contrast, glossing, a system using natural language word labels to represent signs, offers a simpler and expressive approach. It has become a popular choice for sign language translation studies due to its formal structure and understandability.

Figure 1.1: Sample Sign Language Images

Following Arabic, Hausa, and Swahili, Afan Oromo, a Cushitic language within the Afro-Asiatic family, is the fourth most spoken language in Africa. It is primarily spoken by the Oromo people, as well as neighboring communities, and is native to both the Oromia region of Ethiopia and parts of northern Kenya. Afan Oromo is one of the five Ethiopia's official languages, along with Amharic, Tigrigna, Afar and Somali.

Sign Language Translation is a computational task focused on automatically interpreting sign language. In today's digital world, successful sign language translation systems are crucial for bridging the communication gap experienced by deaf and hard-of-hearing individuals.Many sign language translation researches have focused on isolated gestures or facial expressions, neglecting the broader linguistic structure and contextual meaning within complete sentences. Traditional sign language translation methods often suffer from low accuracy, limited scalability, and lengthy processing times.(Mohamed, Hesahm & Ammar, 2021). For a while, machine learning methods have been utilized in sign language recognition. More recently, the emergence of deep learning has encouraged researchers to investigate its effectiveness in recognizing and translating sign languages. Deep learning networks, inspired by biological structures and processes, offer a different approach compared to traditional

methods (Ankita and Parteek, 2020).

Sign language translation is essential for enabling communication between the deaf and hearing communities (Jiangbin et al., 2020). The process of translating between written text and sign gloss can be categorized into three main approaches:

- Converting written text into sign language,
- Translating sign language into written text, and
- Facilitating bi-directional translation between sign language and written text.

1.2 Motivation of the Study

Communication is an essential part of human life, be it with friends, family, co-workers or strangers. Communication with hearing-impaired is a major challenge, as it is done using only body parts or gestures instead of sounds. This gets even more complicated in countries like Ethiopia, as there is no formal education to non-disabled people, at least not at a level of curriculum. Currently, according to WHO report, it is estimated that the number of hearing-impaired people in the world is more than 430million, and more than 5 million in Ethiopia. This figure is expected to grow to 700million worldwide by 2050, which makes the development of a communication mechanism a necessity. Developing a system to address this challenge is inspiring, as it helps bridge the communication gap between individuals with hearing impairments and those without disabilities. Developing translation model for sign languages is also a necessity in many areas of our day to day life, where conveying information for both hearing impaired and non-disabled people, including transportation sectors, religious institutions and commentary for parliamentary meetings among others.

1.3 Statement of the Problem

Although significant research has been conducted on sign language recognition and translation, effectively handling the variation of body poses and hand gestures remains a formidable challenge. This issue is particularly pronounced for Afan Oromo Sign Language, where there is not only a lack of organized datasets but also a noticeable absence of research focusing on capturing dynamic body poses and accurately translating full phrases and sentences. Moreover, the inherent complexity of Afan Oromo's linguistic structure and the contextual nuances of its spoken form further complicate the translation process.

1.4 Research Questions

This study aims to address the following research questions:

RQ1. What deep learning models can be employed for translating Afan Oromo Sign Language into Afan Oromo text?

RQ2. What approaches can be used to enhance the accuracy and efficiency of Afan Oromo Sign Language translation?

RQ3. How can dynamic body poses and hand gestures be handled during the sign language recognition and translation?

1.5 Objective

1.5.1 General objective

Developing a sign language to text translation model for Afan Oromo Sign Language is the general objective of the study.

1.5.2 Specific Objective

The specific objectives that are addressed to achieve the general objective of the study are the following:

- Collecting and organizing dataset for Afan Oromo Sign Language.
- Designing a pre-processing technique to enhance quality of the sign language data set.
- Designing the proposed model so that it handles not only the statics signs and gestures, but also dynamic poses and gestures.
- Taking a through consideration of the linguistic structure of the spoken language.
- Training the proposed deep learning model with the pre-processed dataset.
- Evaluating the model with a test data set using the appropriate evaluation metrics.

1.6 Scope and Limitations of the Study

1.6.1 Scope of the Study

This study aims to address the specified research questions. Given the available time and resources, the scope of this research includes: -

- Preparation and organization of Afan Oromo sign language dataset.

- ➢ Using deep learning approach for encoding-decoding of the pre-processed dataset.
- ➢ Translation of the sign language hand gestures and bodily poses to Afan Oromo text.

1.6.2 Limitation

This research work faced the limitations listed below:

- There is no publicly available dataset for Afan Oromo sign language. One of the limitation for this research is the availability of the organized data.
- The collected data amount, is enough to train the model, but it is not a large and diverse dataset.
- The dataset primarily consisted of short phrases and sentences.

1.7 Significance and Contribution of the Study

1.7.1 Significance of the study

Sign languages are essential for communication within many deaf and hard-of-hearing communities. Recognition and Translation of Sign Language helps in providing efficient communication means between hearing-impaired and normal people. It does so by reducing the communication gap between the signed and spoken language. This in turn helps in decision making, conveying information, and sharing scientific knowledge.

The application domains of AI and Deep-Learning are growing tremendously. Researchers that want to further explore SLT, specially those willing to study Afan Oromo SLT could make use this thesis.

1.7.2 Contribution of the study

The contribution of this research include

- ➢ This research presents significant progress in the translation of Afan Oromo Sign Language phrases and sentences into written Afan Oromo text using deep learning models.
- ➢ The creation of a new dataset by extracting frames from sign language videos.
- ➢ The methodology and findings can serve for future studies on other low-resource sign languages.

- ➢ Presents performance comparison of the encoder-decoder LSTM and Transformer on Afan Oromo sign language to text translation
- ➢ Potential for real-world applications to facilitate the communication among the hearing-impaired community in Afan Oromo speaking regions.

1.8 Organization of the Thesis

Chapter one: This part of the study is used to describe the introductory part of the study which introduces the general idea of this research work like the motivation of the study, problem statement, the objective of the study, research questions, scope, and contribution of the study.

Chapter Two: This chapter provides background information and reviews existing research relevant to sign language translation, including the use of deep learning, LSTM networks, and Transformer models.

Chapter Three: This chapter details the methodology employed to translate Afan Oromo Sign Language into written Afan Oromo. It covers data collection processes, along with the software and hardware tools used in conducting this research.

Chapter Four: This chapter details the architecture proposed for the Afan Oromo Sign Language to Afan Oromo Text translation, and the design of the proposed solution architecture. The architecture proposed in this chapter demonstrates how the encoder-decoder deep learning technique solves the research problems and answers the research questions.

Chapter Five: In this chapter the proposed Afan Oromo sign language to Afan Oromo text translation using LSTM and Transformer encoder-decoder models is implemented. This chapter covers the entire process, from setting up the development environment and preprocessing of the dataset to training procedures and code snippets for the implementation.

Chapter Six: This chapter elaborates results of the implementation of the LSTM and Transformer encoder-decoders. The results are compared using graphs and tables.

Chapter Seven: Summarizes the research work and concludes the results. Recommendations for future work are also presented.

CHAPTER TWO

2. LITERATURE REVIEW

2.1 Conceptual Background

Effective communication is essential for mutual understanding within any community. For the hearing-impaired and people of hard-hearing problems, sign language, a visual and gestural form of communication, plays this crucial role. Sign language differs significantly from both written and spoken languages in its structure and expression(Netsanet, Million and Chala,2021). Sign languages are the primary means of communication within deaf communities worldwide. These languages rely on gestures, hand movements, and facial expressions to convey meaning. (Uzma et al, 2021). Sign language facilitates communication through various hand gestures representing letters, words, or entire sentences. This allows hearing-impaired individuals to express themselves and promotes understanding between them and others. (Ankita and Parteek,2020). Automating sign language translation begins with standardizing its representation. Several notation systems exist, including Stokoe, HamNoSys, SignWriting, and Gloss Notation. However, Stokoe notation lacks facial expressions and body movements, which are crucial elements of non-manual communicatin, limiting its use in accurate translation. HamNoSys, designed for 3D avatar animation, also struggles to represent these non-manual components. SignWriting, while visually descriptive, poses challenges for computer analysis. Glossing, a system that uses natural language words to represent signs, offers a simpler, more expressive, and formal approach, similar in concept to finger-spelling, Morse code and Braille. Because it directly links signs to natural language words, glossing effectively conveys meaning and has become a focus of sign language translation research due to its simplicity, expressiveness, and formal structure (Mohammed, Hesahm and Ammar, 2021).

Sign Language Recognition (SLR) focuses on extracting information from sign language, such as recognizing finger-spelling or classifying signs. Sign Language Translation involves converting information from one language to another, which can include translations between signed languages, between a signed language and a spoken language, or even between text and a signed language. This process works in both directions. Finally, Sign Language Synthesis is a process that uses computer-

generated avatars to create sign language from a given meaning (Mathieu et al, 2023).

Sign language recognition systems commonly employ two primary methods: sensor-based and computer vision-based approaches. Computer vision systems utilize cameras to capture visual data, such as images or videos, for processing. However, the use of high-resolution cameras, while providing detailed visual information, necessitates increased memory capacity and computational resources. This can lead to higher costs and complexity in developing and implementing such systems, particularly due to the advanced image processing techniques and potentially expensive sensors required (Maria et al, 2022).

Deep learning has recently revolutionized natural language processing (particularly neural machine translation) and computer vision (especially image and video captioning). As a result, researchers are now applying these powerful learning methods to the field of sign language understanding (Tejaswini et al, 2021).

2.2 Deep Learning

Deep learning is a type of machine learning inspired by the human brain. It uses multi-layered neural networks to identify complex patterns in data, leading to better predictions and classifications. A key difference from traditional machine learning is that deep learning models learn these patterns automatically from raw data, rather than relying on humans to pre-define them.(Kalideo, Olaoye, 2024).

Deep learning is distinguished by its use of deep neural networks, which are neural networks with many layers of interconnected nodes. These networks are capable of learning complex data representations by identifying hierarchical patterns and features in the data. One of the notable advantages of deep learning is its ability to learn and refine its understanding directly from data, removing the need for manual feature selection. Deep learning is a rapidly developing area within artificial intelligence, utilizing raw data, such as that from neuroimaging, to learn relevant features (S. Roobini, M. S. Kavitha, S. Karthik, 2024). The popular deep learning models are:

1. Convolutional Neural Networks(CNNs): Convolutional Neural Networks (CNNs) are a class of deep learning models that have demonstrated remarkable success in image recognition and processing tasks. They operate by employing multiple layers of convolutions, which are mathematical operations that extract features from the input image. These convolutional layers are typically followed by nonlinear activation functions, such as ReLU (Rectified Linear Unit), which introduce

non-linearity into the model and enable it to learn complex patterns. CNNs are neural networks that use multiple layers of convolutions and activation functions. A key difference from standard neural networks is that CNNs use convolutions, creating local connections instead of fully connected layers. Many filters are used in each layer, and their combined output produces the final result(Hiransha et al, 2018).CNNs are a prominent deep learning algorithm known for their ability to classify inputs through shift-invariant pattern recognition. These networks, typically composed of input, convolutional, pooling, and fully connected layers, are widely used in computer vision due to their specific characteristics (Zhang, 2023). :

a) Convolutional Layer: Convolutional Neural Networks (CNNs) utilize several key layers. First, convolutional layers extract features from input data by filtering it and identifying local patterns. This process makes CNNs well-suited for handling the spatial arrangement of image data.

b) Pooling Layer: Pooling layers reduce the data's dimensionality through downsampling. Common pooling methods, like maximum and average pooling, decrease the model's sensitivity to feature location and improve computational efficiency (Rakrah Singh et al, 2022).

c) Fully Connected Layer: Finally, after multiple convolutional and pooling layers, one or more fully connected layers map the extracted high-dimensional features to target class predictions. These fully connected layer outputs are often passed through an activation function, such as softmax, to produce the final classification.

2. Recurrent Neural Networks(RNNs): In recurrent neural networks(RNNs), the output from a prior step is incorporated as input for the current step. A key characteristic of RNNs is their hidden state, which stores the sequence information. This hidden state, also known as memory state, retains information from prior inputs. RNNs use the same parameters across all inputs or hidden layers, performing the same task and thus reducing parameter complexity compared to other neural networks. In contrast to multilayer perceptrons, RNNs receive input from two sources: the past and the present (Hiransha et al., 2018). In the computation process of RNN, each neuron is used as a memory cell. RNNs are a type of artificial neural networks(ANNs) that the capacity to retain memory. They process sequential or time-series data (ordered data where similar items are grouped) and utilize shared

parameters across the network.

There are four kinds of RNN:

a) One-to-One RNN: A one-to-one RNN functions like a standard neural network, with only one input and one output.

b) One-to-Many RNN: A one-to-many RNN, conversely, takes only one input and produces multiple outputs, as seen in image captioning where a single image input leads to a multi-word sentence output.

c) Many-to-One RNN: A many-to-one RNN processes multiple inputs across different network states but generates only one output. Sentiment analysis, where multiple words are input to predict a single sentiment, exemplifies this type.

d) Many-to-Many RNN: Finally, many-to-many RNNs handle scenarios with multiple inputs and corresponding multiple outputs, such as in language translation.

Even if an RNN model has memory, most of the time it suffers from gradient vanishing and exploding problems.This problem is solved in long-term short-term memory (LSTM). The sequential processing of the recurrent neural network(RNN) is shown in the diagram below.

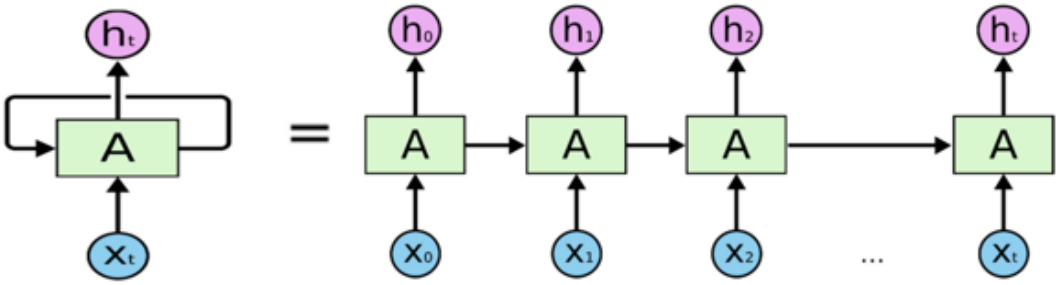


Figure 2.1 RNN processing (Le et al., 2019)

3. Long-Short Term Memory(LSTM): Long short-term memory (LSTM) networks are a specialized form of recurrent neural networks (RNNs) that incorporate mechanisms to mitigate the vanishing gradient problem and capture long-range dependencies in sequential information(Hiransha et al., 2018). This type of artificial neural network employs a series of gates to control the flow of sequential data—how it's inputted, stored, and outputted. In LSTM, the problem in RNN, which is the exploding gradient and vanishing problem, is solved by introducing forget gates, which can remember history by extending their memory in recurrent neural networks

(Siami-Namini et al., 2019). This algorithm can learn from its long-time experience, unlike RNN. Long short-term memories are used to make RNN remember inputs over a long period (Bouktif et al., 2018). The LSTM contains three gates: an input gate, a forget gate, and an output gate. An input gate is used to determine whether a new input is in or not. The second gate is the forgetting gate, which is used to avoid or delete information that is not important. The main task of the forget gate is to determine to what extent to forget the previous historical data. An output gate has to determine what output is to be generated at the current time step. The figure 2.2 illustrates the flow diagram of a Long Short-Term Memory (LSTM) network. Key components shown include the previous cell state (C_{t-1}), the current cell state (C_t), the output from the previous cell (h_{t-1}), and the current cell's output (ht). The diagram also highlights the input gate (it), the forget gate (ft), and the output gate (Ot), which uses a sigmoid function (Hiransha et al., 2018).

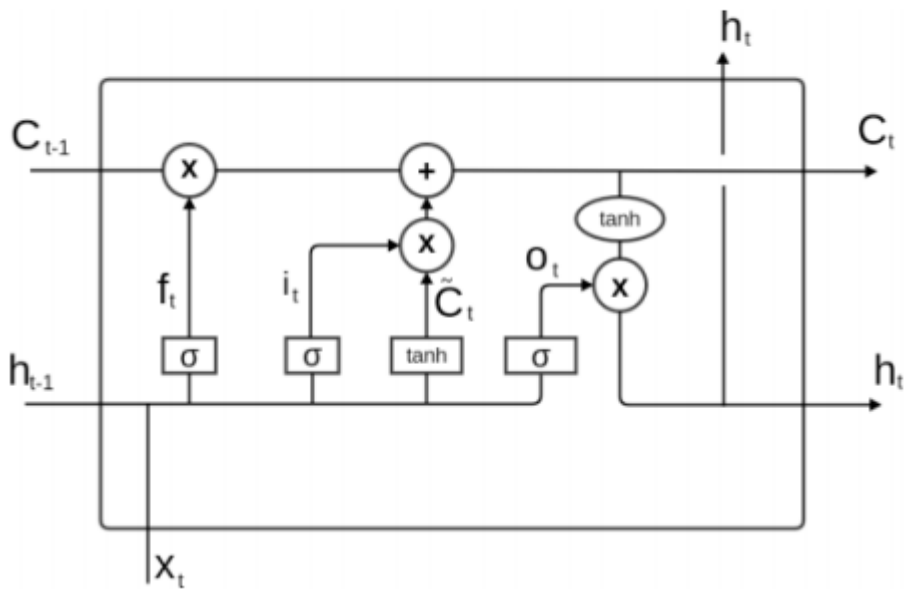


Figure 2.2: Flow diagram of LSTM cell (Hiransha et al., 2018)

4. Auto-Encoders: Autoencoders are deep learning models that utilize neural networks to derive a lower-dimensional, compressed representation of input data, effectively capturing the essential characteristics. They then reconstruct the original input from this compressed version. These networks can be combined to create deeper, hierarchical models capable of organizing, compressing, and extracting high-level features from data, even without labeled training examples (Shuangshuang and Wei, 2023).

The Basic AE: Autoencoders transform an input vector into a compressed "code" vector using learned recognition weights. Generative weights are then used to reconstruct the original input vector from this code (Hinton and Richard, 1994). The architecture of the basic autoencoder consists of three layers: an input layer, a hidden layer (where the code is generated), and an output layer.

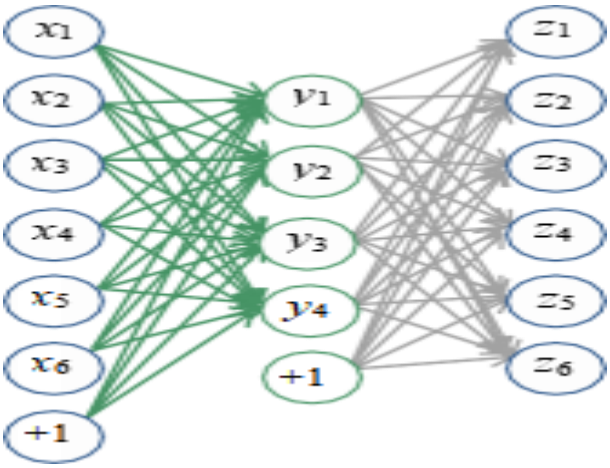


Figure 2.3. Basic AE(Shuangshuang and Wei, 2023).

- **Regularized AEs:** Regularized autoencoders are variants of the basic autoencoders that incorporate different forms of regularization (Guo et al, 2016). In these networks, an encoder function (f) maps inputs (x) to an internal representation (f(x)), which is then mapped back to the input space by a decoder function (g). The regularization process encourages the encoder to discard some information from the input or represent it with reduced precision (Alain and Bangio, 2014).

5. Transformers: Transformer is a neural network architecture that has become a foundation for many natural language processing models. This model does not use recurrence; instead, it uses an attention mechanism to capture global relationships between the input and output (Vaswani et al, 2017). Transformers can allow for significantly more parallelization which makes them much faster to train on large datasets.

- **Vanilla Transformer:** The Transformer model, initially proposed by Vaswani et al. (2017), utilizes an encoder-decoder structure. The encoder processes the input symbol sequence, converting it into a series of continuous, high-level representations. Subsequently, the decoder leverages these encoded representations to generate the output sequence in an iterative

manner, attending to the encoder's output. Both the encoder and decoder are composed of six identical layers. Each of these layers comprises two sub-layers: a multi-head self-attention mechanism and a position-wise feed-forward network. Additionally, the decoder layers incorporate a third sub-layer that performs multi-head attention over the encoder's output.(Vaswani et al, 2017).

- **Variants of Transformer:** These are variations of the original Transformer model have been developed, differing in architecture, pre-training techniques, and applications (Lin et al, 2021).

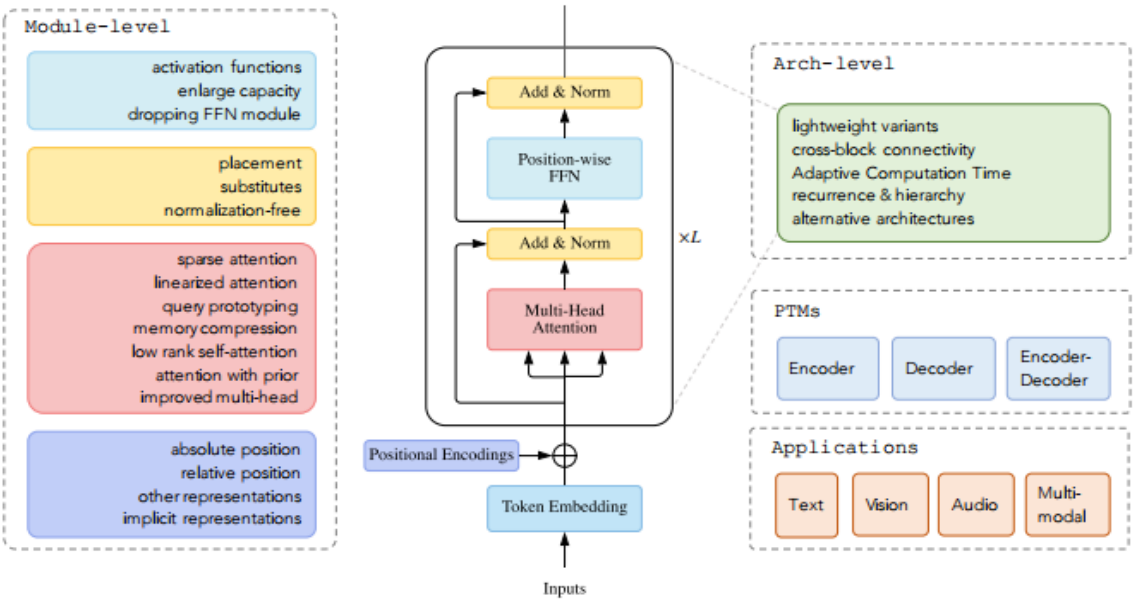


Figure 2.4: Categorization of Transformer variants(Lin et al, 2021)

Transformer Model Usages: Transformers can be utilized in different configurations depending on the task(Lin et al,2021). A complete transformer, employing both encoder and decoder components, is common for sequence-to-sequence mapping tasks like machine translation. Alternatively, using only the encoder is suitable for sequence labeling or classification. Finally, a decoder-only architecture, which omits the cross-attention mechanism, is typically employed for generative sequence tasks such as language modeling.

2.3 Related Works

In 2023, Diriba, Etana, and Hawi employed the YOLOv5 deep learning model to detect and translate Afan Oromo sign language. Their dataset comprised 90 classes of alphabet, number, and word signs, gathered from five special needs instructors. The

model, structured with a CSPDarknet backbone, a PANet neck, and a YOLO layer head, was trained on video frames and outputted Afan Oromo text. Roboflow was utilized for data pre-processing and format conversion. The model's performance was gauged using accuracy, recall, and mean average precision (mAP).

A study in 2024 by Muktar, Aantu, and Chala focused on developing an automatic sign language recognition model for Afan Oromo words using ResNet-50. They compiled a dataset of 25 frequently used Afan Oromo words signed by students and teachers from various institutions. To enhance classification performance, they incorporated segmentation and Gabor filters. The model's effectiveness was measured by recall, precision, and f1-score.

A study by Netsanet, Million, and Chala (2022) explored bridging the communication gap between Amharic speakers with and without hearing impairments. Their research focused on creating a digital text converter for Amharic using Ethiopian Sign Language. They employed a combination of deep learning models: EfficientNetB0, ResNet50, and LSTM. The LSTM component was specifically used for extracting sequential information from image frames. Their dataset consisted of 2745 images extracted from sign language videos sourced from Addis Ababa University. The system's performance was measured using accuracy, precision, and the F1-score.

In 2015, Mary Elizabeth investigated machine translation for low-resource languages, including sign languages, which often lack substantial parallel corpora. Elizabeth utilized Statistical Machine Translation with the Moses Toolkit to translate English text into American Sign Language (ASL) gloss. A limited parallel corpus of English and ASL gloss, provided by The Church of Jesus Christ of Latter-day Saints, served as the dataset. The model's evaluation involved 5-fold cross-validation and the BLEU score.

More recently, Uzma, Mohd, and Adanan (2023) tackled the challenge of English to Pakistan Sign Language (PSL) translation using deep learning. Their work aimed to improve upon the limited accuracy previously observed in Neural Machine Translation models applied to spoken-to-signed language translation. They implemented an RNN-based Neural Machine Translation model incorporating Bahdanau attention and GloVe embeddings. The model architecture featured a multi-stack RNN encoder-decoder structure. A dataset of 50,000 sentences in both PSL and English was used. Performance was assessed using both BLEU and Word Error Rate (WER) metrics.

Avola et al. (2018) employed a deep learning model using Recurrent Neural Networks (RNNs), specifically a stacked Long Short-Term Memory (LSTM) network, for hand gesture recognition. Feature extraction was performed using finger angle data acquired from a Leap Motion Controller, and the model was evaluated on the ASL and SHREC datasets.

More recently, Mohammed, Hesahm, and Ammar (2021) aimed to address the limitations of traditional sign language translation by incorporating linguistic structure and context. They utilized deep learning models, including Gated Recurrent Units (GRUs) and LSTMs, enhanced with Bahdanau and Luong attention mechanisms for bidirectional translation between signed and spoken languages. Their proposed Neural Machine Translation system involved preprocessing and encoding-decoding phases, with attention models connecting the encoder and decoder RNNs. The ASLG-PC12 and Phoenix-2014T corpora served as datasets, and performance was assessed using BLEU and ROUGE scores.

Camgoz et al. (2020) developed a transformer-based architecture for simultaneous continuous sign language recognition and translation. A Connectionist Temporal Classification (CTC) loss function was implemented to unify both tasks within a single architecture. Their sign language transformers consisted of 512 hidden units and 8 attention heads per layer, and the PHOENIX14T corpus was used. Word Error Rate (WER) and BLEU scores were the evaluation metrics for recognition and translation, respectively.

Building upon this work, Cabot, Tarres, and Giro-I-Nieto (2022) reproduced the transformer-based approach of Camgoz et al. (2020) using the multimodal American Sign Language How2Sign dataset. They established baseline recognition and translation results, introducing a novel sentence-based alignment for the How2Sign videos. WER and BLEU scores were again used for performance evaluation.

The table below represents summary of related works.

Table 2.1: Summary of related works

Title of the Article	About the Research and The Research Problem	Methodology	Research Gap

Signed Language Translation into Afaan Oromo Text Using Deep-Learning Approach (Diriba Negash,Etana Fikadu,Hawi Fikadu) (2023)	About: The primary focus of the paper is the translation of Signed Afan Oromoo to Afan Oromo text. Research Problem: Detection of Sign Language for Afan Oromo	• Model trained using transfer learning techniques from YOLOv5 and CNN • Manually gathered dataset • Accuracy,recall and mAP	No consideration of pose or gesture effects No usage of phrases or sentences while training the model
Developing Sign Language Recognitiom Model for Afan Oromo Words Using a Deep Learning Technique(Muktar Bedaso. Antu Hussein, Chala Dirriba, 2004)	About: This study seeks to improve educational accessibility for deaf individuals. Research Problem: The absence of Afan Oromo sign language recognition systems	• ResNet50 with Segmentation and Gabor filter • Manually gathered dataset • Recall, precision and f1-score	No consideration of pose or gesture effects No usage of phrases or sentences while training the model No numerical results for evaluation metrics.
A Generic Approach Towards Amharic Sign Language Recognition(Netsanet Yigzaw,Million Meshesha,Chala Diriba) (2022)	About: This study aims to create a system that translates Amharic digital text into Ethiopian Sign Language. Research Problem: Communication gap between the deaf and non-disabled people by using Amharic Sign Language recognition	• Hybrid of LSTM and CNN for sequence extraction from image sequence • EfficientNetB0 and ResNet50 to extract features from single image frame • Sign Language in video format from AAU • Accuracy, Precision and f1-score	• No consideration of pose or gesture effects.

English to American Sign Language Gloss Translation(Mary Elizabeth Bonham) (2015)	About: This thesis focuses on the creation and assessment of a machine translation system that converts English text into American Sign Language gloss. Research Problem: major obstacle in machine translation, particularly for languages with low-resource such as sign languages, is the scarcity of extensive parallel data.	• Statistical MT approach • Mooses Translation Toolkit • A parallel corpus of small size, containing English text paired with ASL glosses, obtained from The Church of Jesus Christ of Latter-day Saints. • 5-fold cross validation with BLEU score	• Using only a single evaluation technique , and failed to mention the reason for their choice.
Exploiting Recurrent Neural Networks and Leap Motion Controller for Sign Language and Semaphoric Gesture Recognition(Danilo Aviola, et al,2018)	About: This work utilizes the angles between finger bones as features to train a Recurrent Neural Network. Research Problem: Recognition of different hand gestures for Sign Languages using skeletal-based modeling.	• RNN model • Leap Motion Controller (LMC) sensor for data acquisition • Gestures defined by ASL and SHREC data set for testing • Accuracy, precision, recall and f-score	• The thesis focuses only on hand gestures. • Does not account for dynamic poses.

CHAPTER THREE

3. RESEARCH METHODOLOGY

3.1 Overview

This chapter details the methodology employed for translating Afan Oromo Sign Language into Afan Oromo text. It covers the data collection process, along with the software and hardware tools utilized in the research.

Figure 3.1 presents the methodology used in this study, from dataset collection to the evaluation of the model.

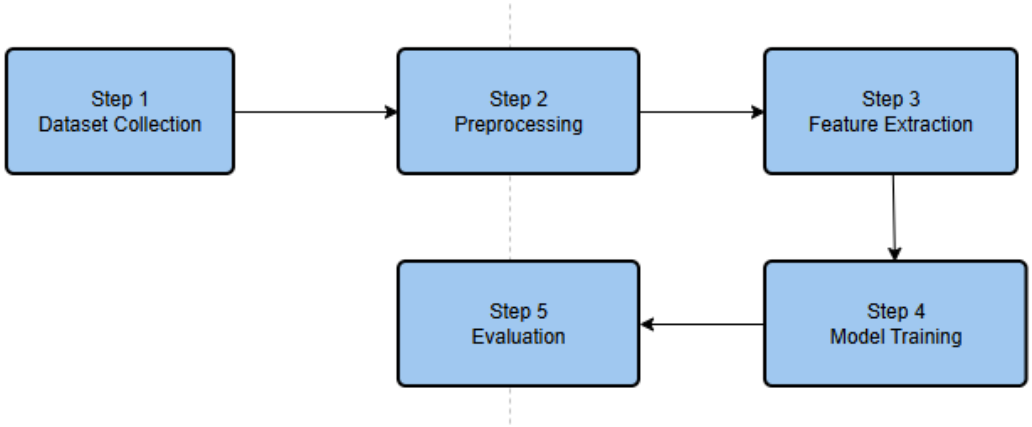


Figure 3.1 Research Methodology

3.2 Datasets collection

Deep learning models, subset of machine learning, are fundamentally reliant on data for training. In applications like sign language translation using these models, data acquisition and preparation are crucial initial steps. A significant challenge is the lack of a publicly available dataset for Afan Oromo Sign Language. This research addresses this gap by collecting two types of data: one for training the deep learning model and the other for evaluating its performance.

The Afan Oromo Sign Language dataset used for this thesis will be the frames extracted from collection of a videos streams that are collected form different individuals with knowledge of Afan Oromo Sign Language . The video data is collected by recording four different signers using different hand gestures and body poses. Different sentences and phrases are recorded using digital camera.

3.3 Evaluation Methods

Evaluation Metrics, known as Quantitative measures, are essential for determining the performance and effectiveness of statistical or machine learning models. They give significant insights into model accuracy and facilitate comparisons between different models or algorithms. While qualitative analysis methods exist, quantitative metrics are specifically employed to assess Sign Language Translation models. In this study word error rate(WER), and ROUGE-score will be used for evaluation of the proposed model.

A. **Word Error Rate(WER):** The Word Error Rate (WER) quantifies the difference between a machine translation system's output and the actual, correct text. It is calculated as follows:

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

Where: S-- number of substitutions
I-- number of insertions
D-- number of deletions
N-- number of words in the reference

ROUGE(Recall-Oriented Understanding for Gisting Evaluation): ROUGE is a set of metrics commonly used to assess the performance of systems designed for automatic text summarization and machine translation. It functions by comparing the output of these systems (summaries or translations) with one or more human-created reference texts.

ROUGE focuses on quantifying the similarity between the machine generated text and the reference text, specifically by examining the extent of shared words, phrases, and word order. Two of the most frequently employed ROUGE metrics are described below.

- ROUGE-N: quantifies the n-gram overlap between these texts.

$$ROUGE - N = \frac{\sum_{s \in Re\ fsumm} \sum_{n-gram \in S} Countmatch(n-gram)}{\sum_{s \in Re\ fsumm} \sum_{n-gram \in S} Count(n-gram)}$$

- ROUGE-L: evaluates the longest common subsequences (LCS) shared between generated and reference texts, taking into account sentence structure and word order.

$$ROUGE - L = \frac{(1+\beta^2).\Pr ecision.\mathrm{Re}\, call}{\beta^2.\Pr ecision + \mathrm{Re}\, call}$$

3.3. Development Tools

This research employed a variety of tools for the design and implementation of the proposed system. These included tools and platforms for prototype development, UML modeling software, and other resources relevant to the research. The following sections provide further details on these tools.

3.3.1 Design Tools

Software tools that facilitate the design of architectures and infrastructure for machine learning model development and deployment are known as design tools. This research utilizes Enterprise Architect, a visual modeling and design tool compliant with OMG UML, for system design. Enterprise Architect's capabilities include software system design and construction, business process modeling, and domain-specific modeling.

3.3.2 Hardware tools

The following hardware tools are used in the research implementation

Table 3.1: Hardware tools

No.	Tools	Specification	Used for
1.	GPU	GeForce RTX 2070 Super 6 GB RAM	Enhancing computational resources and improving training efficiency
2.	HDD	WD 1TB USB 3.2 Gen 1	Storing the sign language data set

3	RAM	16GB RAM	For performance and speed

3.3.3 Software tools

The implementation phase of this research utilized various software development tools, including integrated development environments (IDEs) and code editors. These tools are detailed below.

Table 3.2: Software tools

No	Software tools	Version	Description
1.	Python	3.12.1	Python is a versatile, high-level programming language that is widely used. It features dynamic typing, automatic memory management (garbage collection), and supports various programming paradigms such as structured, object-oriented, and functional approaches.
2.	Jupyter Notebook	7.0.6	Jupyter Notebook is a widely used and convenient development environment for Python in the fields of artificial intelligence and deep learning.
3.	Scikit-learn	1.3.2	Scikit-learn is an open-source Python library widely used for machine learning. It provides implementations of numerous algorithms for classification, regression, and clustering, and seamlessly integrates with the NumPy and SciPy libraries for numerical and scientific computing in Python.
4.	Anaconda	2022.10	Anaconda is a software package that includes the Python and R programming languages, specifically designed for use in scientific computing. It simplifies the processes of package management and deployment.

5.	TensorFlow	2.1	TensorFlow, an open-source library, is designed for high-performance numerical computation and processing. Its modular design contributes to ease of use.
6.	Keras	3.0.2	Keras is a Python-based API that offers a streamlined approach to constructing and training deep learning models. It functions as an interface for the TensorFlow platform.

CHAPTER FOUR

4. DESIGNING OF THE PROPOSED SOLUTION

4.1 Overview

This chapter details the proposed Afan Oromo Sign Language to Afan Oromo Text translation system, including its architectural design. The proposed model's architecture demonstrates how the encoder-decoder deep learning approach addresses the research questions and problems outlined in Chapter One, and how it bridges the gaps identified in the literature review.

4.2 Architecture of the Proposed Solution

Encoder-decoder LSTM or deep RNN and Transformer are the proposed models for Afan Oromo Sign Language to Text translation. The proposed solution starts with data acquisition, where different sentences of Afan Oromo in signed format are recorded using digital camera. Then image frames are extracted from the recorded video, followed by preprocessing which involves noise removal, skew correction and normalization. The preprocessed data is then split into training set, validation set and testing set using 75%:15%:15% split. Then the encoder-decoder models are fed with the training data, and the model's performance is tested using the test data set. Performance metrics WER and ROUGE-score are used to test the translation outcome.

The following diagram shows architecture of the proposed system.

Figure4.1: Architecture of the proposed system

The work flow of the proposed architecture is also presented as follows:

Figure4.2: Workflow of the Proposed Architecture

4.3 Data Collection

The video data for this study was recorded using a digital camera. A special needs teacher from Shambu College of Teachers Education recorded and sent us fifteen classes of frequently used Afan Oromo sign language words. After extensive training,

four people signed the Afan Oromo phrases and sentences used in this study. Total of thirteen phrases and two sentences were signed and captured in MP4 video format using the same background. A total of of 1678 image frames were collected for the fifteen classes of phrases and sentences. The images are in JPG format, and normalized to 250 x 250 pixels. During feature extraction, the image frames were resized to 224 x 224 pixels, which is the standard input size for feature extraction models like ResNet50 and VGG16. The table below presents the phrases and sentences used for preparing the dataset.

Table 4.1 Class of Afan Oromo Sign Language Dataset

No	Afan Oromo Phrase or Sentence	Number of extracted frames
1	Akka Aaruuf	89
2	Akka Tasaa	87
3	Galgala Gaarii	98
4	Ganama Gaarii	104
5	Gara Fulduraatti	91
6	Garaa Qulqulluu	96
7	Guyyaa Gaarii	105
8	Hafuura Qulqulluu	92
9	Irraa Eeguu	112
10	Irraa Hafaa	108
11	Irraa Kaasuu	99
12	Maqaan Kee Eenyu	148
13	Nama Gaarii	146
14	Nama Hamaa	149
15	Ani Sin Jaaladha	154
Total Number of Extracted Frames		1678

4.4 Architecture of the Data Preprocessing

The data acquired using digital camera contains numerous noises, needs an enhancement, and should be normalized. Preprocessing is an operation performed on the original data table to transform it into new table. The aim of data preprocessing is making ready the file to be used for training and testing the encoder-decoder model.

The data preprocessing techniques used in this study are the followings:

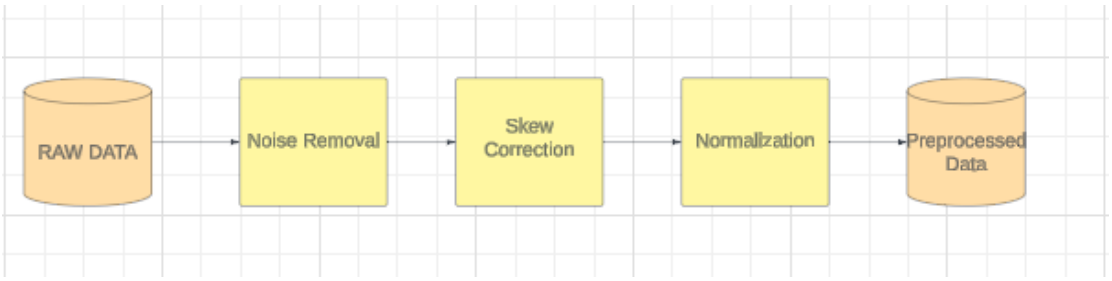


Figure 4.3 Preprocessing Architecture

Image Enhancement Techniques: These techniques refine the image so that the resulting output is of higher quality than the original, providing a better input for automated processing systems.

Noise Removal: This step targets the elimination of unwanted or random variations in pixel values, which can otherwise degrade image clarity and overall quality.

Normalization: Normalization is the process of adjusting pixel intensity values so that they conform to a specified range. This step ensures that images exhibit consistent brightness and contrast, which is crucial for accurate analysis. By rescaling the pixel values, the image is made uniform, improving the reliability of subsequent processing steps.

Skew Correction and Thinning: The skew angle is determined by finding the peak in the histogram of gradient orientations from the grayscale image. Deskewing then corrects this misalignment by rotating the image by an equivalent angle in the opposite direction. Additionally, thinning procedures may be applied to further streamline the image features.

4.5 Feature Extraction

In deep learning and data analysis, feature extraction plays a crucial role in identifying and isolating pertinent characteristics from raw data. This process primarily serves to minimize the computational resources needed to represent extensive datasets (Bishop & C.M, 2006).

Feature extraction helps in dimensionality reduction by providing more structured representation of data. This enhances the performance of the deep learning models.

4.5.1 ResNet50

ResNet50 is a deep convolutional neural network(CNN) that is used primarily for image classification. It is a variant of Residual Network(ResNet) family known for its ability to learn complex features.

4.6 LSTM Framework

LSTM or deep RNN is an improved version of RNN that excels in sequence prediction tasks. The key components of LSTM are:

- Memory Cell(ct): In an LSTM, the memory cell serves as a crucial element that retains information across time. It maintains flow of information, and it can add or remove information as required.
- Forget Gate(ft): In an LSTM network, the forget gate plays a crucial role in filtering out outdated information from the previous cell state. Operating via a sigmoid layer, it takes the preceding hidden state (h_{t-1}) and the current input (x_t) to generate a value between 0 and 1 for every cell state element. A 0 means that the specific data should be entirely discarded, while a 1 implies it should be completely preserved.

$$ft = \sigma(Wo \cdot [ht-1, xt] + bo)$$

- Input Gate(it): The input gate controls the addition of new data to the memory cell. It has two components.
  - Input Gate(it): sigmoid layer deciding which values are to be updated.

$$it = \sigma(Wi \cdot [ht-1, xt] + bi)$$

  - Candidate Memory (ĉt): a tanh layer creating a vector of new candidate values to be added to the cell state.

$$\hat{c}t = tanh(Wc \cdot [ht-1, xt] + bc)$$

- Output Gate(ot) and Hidden State(ht): The output gate controls the flow of information from the cell state to both the next hidden state and the cell's output.
  - Output Gate(ot): sigmoid layer deciding which parts of the cell state to output.

$$ot = \sigma(Wo \cdot [ht-1, xt] + bo)$$

  - Hidden State(ht): represents a filtered version of the cell state. It is obtained by applying the tanh activation function to the cell state and then multiplying the result by the output gate.

$$ht = ot \cdot tanh(ct)$$

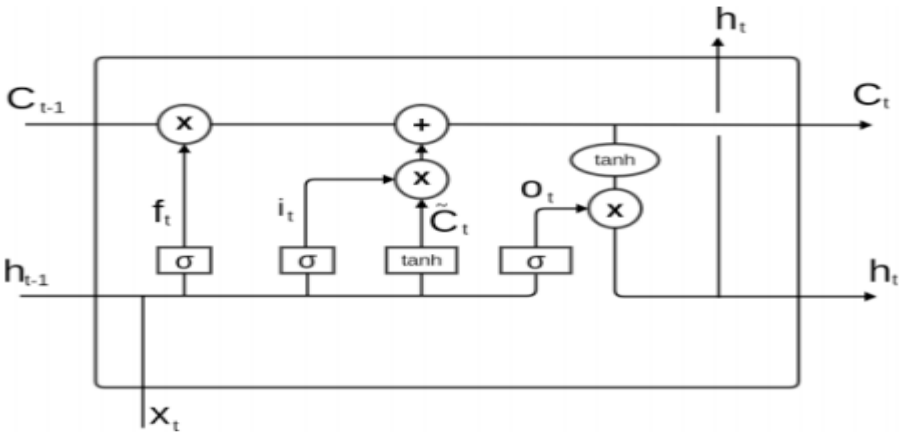


Figure4.4 Flow diagram of LSTM cell (Hiransha et al., 2018)

4.7 Encoder-Decoder LSTM Architecture

A Long Short-Term Memory (LSTM) based encoder-decoder architecture offers a flexible and effective approach for sequence-to-sequence mapping problems. The use of LSTMs in both the encoder and decoder components enables the model to learn complex temporal relationships and produce precise output sequences.

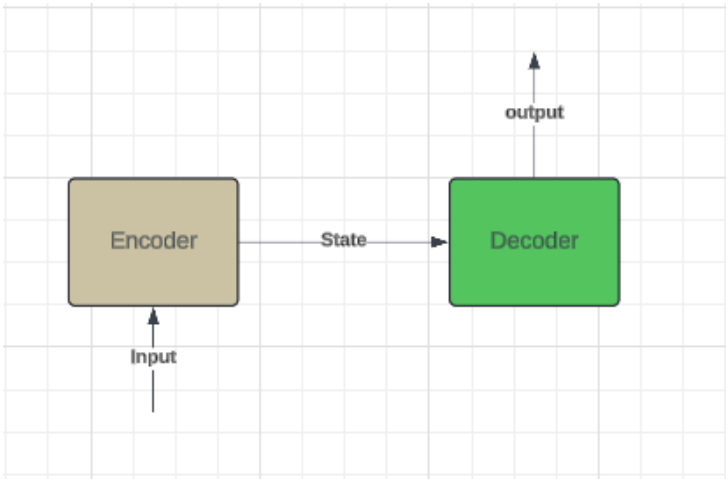


Figure4.5: Encoder-Decoder Architecture

4.7.1 Encoder LSTM Architecture

The Encoder will process the input sequence and compress it into a fixed-length context vector(hk, ck).

- Input Sequence: The input sequence can be of variable length, in this case a signed Afan Oromo sentence.
- LSTM Layers: The LSTM processes the input sequence one element at a

time and generates a hidden state at each time step. The hidden states contain information about the sequence up to that point.

- Final Hidden and Cell States: After processing the entire input sequence, the final hidden state of the LSTM serves as the context vector(hk, ck). This vector is meant to encapsulate the meaning of the input sequence.

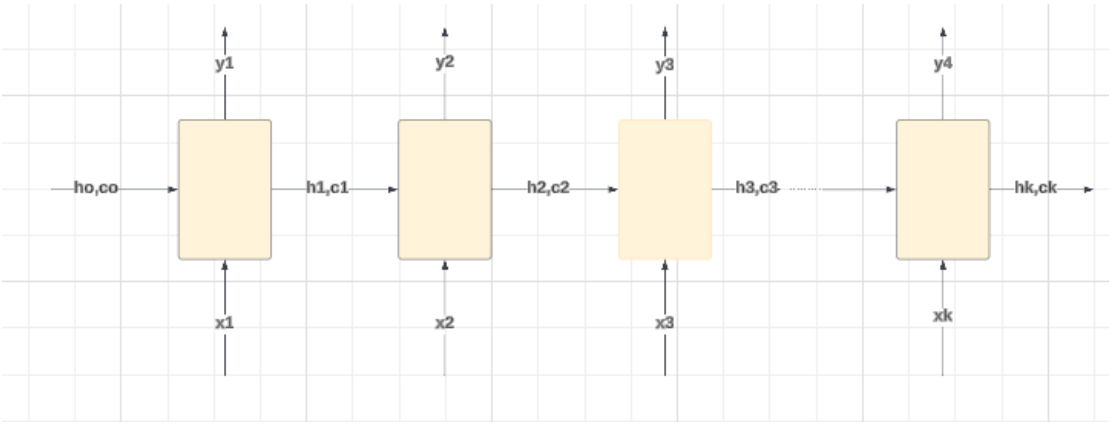


Figure4.6: Architecture of the LSTM Encoder

The encoder processes the input image frame sequence, retaining the LSTM network's internal states (hk, ck) produced at the final time step. While the encoder generates outputs (yi) at each time step during sequence processing, these intermediate outputs are not utilized after the full sequence has been read.

4.7.2 Decoder LSTM Architecture

The goal of the decoder is to generate a caption or text in sequence that matches the Afan Oromo sigh language based the context vector(s) provided by the encoder LSTM.

- Initial State: The Decoder's initial hidden and cell states are usually initialized with the context vector from the Encoder. This context vector delivers a condensed representation of the input sequence to the Decoder.
- LSTM Layers: The Decoder's LSTM processes each step of the output sequence, using its own hidden state and the previous output.
- Output Layer: At each time step, the LSTM's hidden state is used to generate a probability distribution over the set of possible output elements by passing it through a dense layer with a softmax activation function.

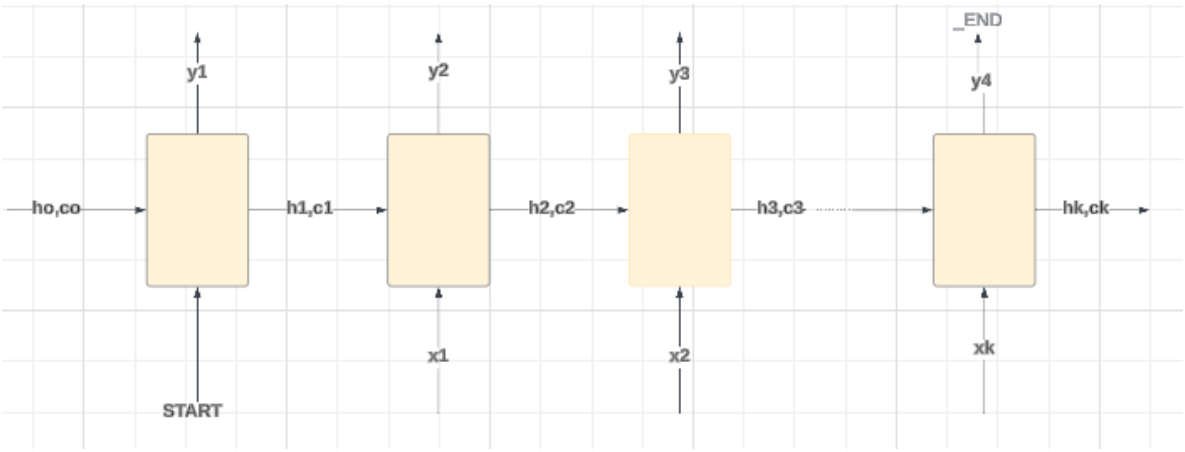


Figure 4.7: Architecture of the LSTM Decoder

The decoder's initial state (h_0, c_0) is initialized with the encoder's final state (hk, ck). This is because the decoder is trained to begin output generation based on the information processed and encoded by the encoder.

4.8 Transformer Encoder-Decoder Architecture

Transformer is a robust deep learning model used for sequence to sequence prediction tasks such as machine translation, modeling, and tasks in computer vision. Transformer is a model that relies entirely on self-attention, without using sequence-alligned RNNs or convolution.

4.8.1 Components of Transformer Framework

In language translation tasks, the transformer is generally implemented as a stack of encoder and decoder layers.

- **Encoder**: The encoder's role is to create a continuous representation from the input sequence. It is composed of L identical layers arranged one on top of the other. In each encoder layer, there is a multi-head self-attention mechanism paired with a position-wise feed-forward network (Lin et al., 2021). Additionally, each module is wrapped with residual connections and layer normalization to support the construction of deeper models.
- **Decoder**: The decoder builds the output sequence one step at a time, drawing on the encoder’s outputs along with its own previously produced tokens. Mirroring the encoder’s design, the decoder is structured into L identical layers. Each of these layers is subdivided into three components: a multi-head self-attention mechanism, a position-wise feed-forward network, and an additional multi-head attention module that focuses on the encoder’s outputs—this is known as cross-

attention (Vaswani et al., 2017).

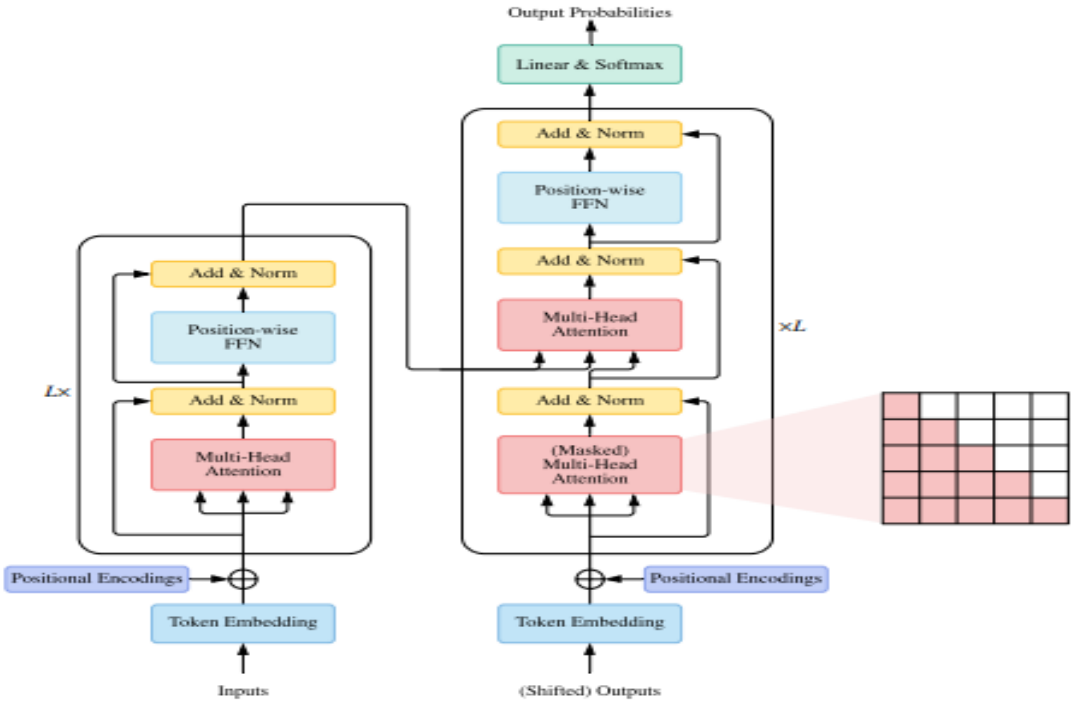


Figure 4.8: Architecture of the Vanilla Transformer(Lin et al,2021)

4.8.2 Attention Modules

Attention mechanisms prioritize key information during processing, while still considering supplementary details. Rather than processing all input elements uniformly, these modules selectively concentrate on the most relevant components for a given task. This approach facilitates the modeling of long-range dependencies, supports parallel computation, and enhances the model's ability to learn contextual relationships within the data.

The attention mechanism functions by transforming a query alongside a set of key–value pairs into an output, where the query, keys, values, and output are all expressed as vectors. The output is produced by taking a weighted sum of the values, with the weights determined by a compatibility function that evaluates the relationship between the query and each key (Vaswani et al., 2017).

Attention(Q,K,V)

Scaled Dot-Product Attention: This mechanism computes attention weights by taking the dot product of the query and key vectors. These dot products are then scaled by the key vector dimension before applying a softmax function, which converts the

scaled values into normalized weights. Finally, the computed weights are used to form a weighted sum of the value vectors.

$$\text{Attention}(Q, K, V) = \text{Softmax}(\frac{QK^T}{\sqrt{d_k}})V = AV$$

Where $A = \text{Softmax}(\frac{QK^T}{\sqrt{d_k}})$ is called the attention matrix, and d_k represents dimensionality of the keys.

Transformers utilize three distinct attention modules, differentiated by the origin of their queries and key-value pairs..

- Self-Attention(Intra-Attention): Self-attention allows to evaluate relationships between all elements in the same sequence,that is, each position in an encoder or decoder layer can attend to all positions in the previous layer of the encoder or decoder, respectively(Vaswani et al,2017)
- Masked Self-Attention: Masked self attention is used only when the model must only attend to prior information.In this, key-value pairs have already been generated, which prevents
- Cross Attention: When a model's focus must shift to a sequence other than its own input, cross-attention is used. In this mechanism, the decoder's previous layer outputs generate the query projections, while the encoder's outputs are used to create the key and value projections (Lin et al., 2021).

4.8.3 Position-wise Feed Forward Network

As described by Vaswani et al. (2017), the position-wise Feed-Forward Network (FFN) is a fully connected module that processes each position independently and in the same manner. It comprises two linear transformations with a Rectified Linear Unit (ReLU) activation function applied between them. This FFN is applied to every position individually and identically.

$$FFN(H') = ReLU(H'W1 + b1)W2 + b2$$

Where H' is the output of the previous layer, and W1 ε RDm x Df, W2 ε RDf x Dm, b1 ε RDf, b2 ε RDm are trainable parameters. Typically Df of the FFN is set to be larger than Dm.

4.8.4 Positional Encodings

Transformers, lacking both recurrence and convolution, do not inherently process

sequential order. Therefore, Positional encodings are added to the input embeddings at the foundation of both the encoder and decoder stacks to convey structural information about the sequence. These encodings are based on sinusoidal functions..

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right)$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right)$$

Where pos is the position and i is the dimension. The positional encodings have the same dimension dmodel as the embedding.

4.8.5 Residual Connection and Normalization

Transformers utilize residual connections around each module, followed by layer normalization (Lin et al., 2021), to facilitate the training of deep models. These techniques contribute to both training stability and speed.

4.9 Hyperparameter Optimization

In deep learning, hyperparameters are pre-set values that govern the training process. Unlike parameters learned during training, these values are manually defined before the algorithm is applied to the data to guide the learning process (Das et al, 2024).

The process of tuning a model's performance on a specific dataset by selecting the optimal configuration of its hyperparameters is known as hyperparameter optimization. In addition to performance enhancing, hyperparameter optimization is crucial for efficient resource utilization and ensuring stability of the model. The followings are hyperparameters to be considered in this study.

4.9.1 Learning Rate

The learning rate determines the magnitude of weight adjustments in a neural network during each optimization step, effectively controlling the pace of model training..

Lower learning rate leads to more model stability and the typical range for learning rate is “0.001” to “0.01.”

4.9.2 Batch Size

The batch size defines how many data samples are used to adjust the model's internal parameters during training. Commonly used batch sizes include 16, 32, 64, and 128.

4.9.3 Dropout Rate

Dropout rate describes the fraction of units that are randomly dropped during training

of the model to prevent overfitting. Higher dropout rates reduce overfitting but can slow the learning process. The typical range for dropout rate is “0.2” to “0.5.”

4.9.4 Epochs

In deep learning model training, an epoch represents one complete pass through the entire training dataset. It describes a full cycle of training on all available data. More epochs give the the model higher opportunities to learn if combined with an early stopping. The typical range for epoch is “10” to “100.”

4.9.5 Number of Hidden Units

This describes the number of memory cells or hidden units present in each LSTM layer. The more hidden units in the network, the more complex patterns the model will capture. But this comes at a cost of overfitting.

4.9.6 Number of LSTM Layers

This describes the number of stacked LSTM layers in encoder-decoder network. In this study we will be experimenting with configuration of encoder-decoder architecture with “2” layer of LSTM, one in the encoder and one for the decoder.

4.9.7 Number of Transformer Heads

This describes the number of scaled dot-product attentions computed independently. Using multi-head attention helps in improved learning as each head can focus on different aspects of the input sequence simultaneously. For the purpose of our study, we will be using 2-head and 8-head transformers.

4.9.8 Optimizer

Optimizers are algorithms used to update the weights of the deep learning network during the training. They maximize efficiency of the model by minimizing the loss fuction. Some of the optimizers for LSTM encoder-decoder model are:

- Stochastic Gradient Descent(SGD): SGD is a fundamental optimization algorithm that employs a single, user-specified learning rate for all parameter updates. This global learning rate remains constant throughout the optimization process. While straightforward to implement and requiring minimal memory compared to more sophisticated optimizers, SGD's uniform learning rate application across all parameters can lead to slow convergence. The range of learning rate for SGD is typically between “0.01” to “0.1”
- Adaptive Moment Estimation(Adam): Adam is a great default optimizer for LSTM encoder-decoder models, especially for tasks with noisy

gradients or sparse data. Its advantages include adaptive learning rate, momentum to accelerate the optimizer in a relevant direction, and faster convergence than vanilla SGD. Adam's default learning rate is "0.001" often tuned between "0.0001" to "0.01."

- Root Mean Square Propagation(RMSprop): RMSprop adjusts the learning rate on a per-parameter basis. It accomplishes this by reducing the global learning rate based on a moving average of the recently observed gradient magnitudes. Its advantages include handling non-stationary objectives, and leads to more stability than vanilla SGD. The default learning rate for RMSprop is "0.001" often tuned between "0.0001" to "0.01."

4.9.9 Activation Function

Activation function is a mathematical function applied to the output of a neuron in a deep learning network. It introduces non-linearity into the deep learning model, making the network to learn and represent complex patterns in the dataset.

Activation functions are important in determining how the dataset is transformed in the deep learning model. The common activation functions for encoder-decoder are the followings.

- Hyperbolic Tangent(tanh): is an activation function that ensures the values in the cell state are always within a reasonable range (-1 to 1). This makes it easier for the network to learn. Convergence during training and faster learning are the advantages of tanh function.

  The formula for tanh activation function is:

$$tanh(x) = e^{x} - e^{-x} \div e^{x} + e^{-x}$$

- Rectified Linear Unit(ReLU): is an activation function that returns the input itself if it receives any positive value, otherwise returns "0" if it receives any negative value. ReLU is generally used in feed-forward layers following LSTM layers, or in non-recurrent layers of deep neural networks. Its advantage are computational efficiency and the introduction of sparsity in the network.

  The formula for ReLU activation function is:

  $$ReLU(x) = \max(0,x)$$

- Leaky ReLU: is a variation of ReLU where the negative input section

allows a small gradient instead of being zero. Leaky ReLU ensures that neurons receiving negative inputs do not become inactive, and improves model learning by allowing gradient flow even for negative inputs.

The formula for Leaky ReLU activation function is:

$$\text{LeakyReLU}(x) = \max(\alpha x, x),$$

where α is a small constant, usually "0.01."

CHAPTER FIVE

5. IMPLEMENTATION OF THE PROPOSED SOLUTION

5.1 Overview

This chapter presents the implementation of the proposed Afan Oromo sign language to Afan Oromo text translation using LSTM and Transformer encoder-decoder models. It encompasses the implementation process, from setting up the development environment and preprocessing the data to the training procedures and illustrative code examples.

5.2 Working Environment

This section describes the hardware environment employed during the implementation of the proposed solution.

Desktop:

- Operating System: MS Windows 10 Pro
- Processor: Intel(R) Core (TM) i5-4590 CPU @ 3.30GHz 3.30 GHz
- Installed Memory(RAM): 12.00 GB
- Graphics Processing Unit(GPU): NVIDIA GeForce GTX 1050 super intalled
- Model: DELL OptiPlex 7020
- System Type: 64-bit OS, x64-based processor

Laptop:

- Operating System: MS Windows 11 Home Single Language
- Processor: 11th Gen Intel(R) Core (TM) i5-1135G7 @ 2.40GHz, 2401Mhz, 4 core(s), 8 logical processor(s)
- Installed Memory(RAM): 16.00 GB
- Graphics Processing Unit(GPU): Intel(R) Iris® Xe Graphics
- Model: Hp Laptop 14s-dq2xxx
- System Type: 64-bit operating system, x64-based processor

5.3 Setup Environments

This section describes the programming tools, online tools, and integrated development environments used in this study.

JupyterLab: JupyterLab provides a web-based, interactive platform for development in scientific computing, artificial intelligence, machine learning, and deep learning. As a successor to the Jupyter Notebook interface, JupyterLab offers enhanced

flexibility and extensibility for working with notebooks, code, and data.

Anaconda: Anaconda is a software package that includes the Python and R programming languages, specifically designed for use in scientific computing. It simplifies the processes of package management and deployment.

Google Colab: Google Colab provides a no-cost, cloud-based environment for developing and running Python code within a browser-based notebook interface. It provides free access to GPUs and TPUs, and uses the Jupyter Notebook interface making it easier to prototype, test, and share different codes.

Keras: Keras is a Python-based API that offers a streamlined approach to constructing and training deep learning models. It functions as an interface for the TensorFlow platform.

TensorFlow: TensorFlow, an open-source library, is designed for high-performance numerical computation and processing. Its modular design contributes to ease of use.

5.4 Training Procedure

The following are steps taken in the training procedure to achieve the goal of the study.

- Dataset Collection
- Data Preprocessing
- Configuring the dataset and GPU
- Building the model
- Compiling the model
- Training the model to implement the proposed solution

5.5 Implementation of Data Preprocessing

5.5.1 Dataset Preparation

After the sign language video was recorded using digital camera, image frames were extracted from the video dataset and saved in separate folders, according to the phrases or sentences they presented, along with their annotated labels. Accordingly, the dataset contained 15 folders for the extracted image frames and their annotations. The code snippet below presents the frame extraction for this study.

```
frame_count = 0
frame_rate = 5

while True:
    ret, frame = cap.read()
    if not ret:
        break

    if frame_count % frame_rate == 0:
        frame_filename = os.path.join(output_dir, f'frame_{frame_count:04d}.jpg')
        cv2.imwrite(frame_filename, frame)

    frame_count += 1
```

Snippet Code 5.1: Frame extraction

For the extracted frames, a data label was created using an online CVAT data annotator, and the annotations were exported as PASCAL VOC .xml files.

The dataset was the split to train,validation and test sets, and moved to their respective folders using the code snippet below.

```
train_images, test_images, train_xmls, test_xmls = train_test_split(
    image_paths, xml_paths, test_size=0.3, random_state=42)

val_images, test_images, val_xmls, test_xmls = train_test_split(
    test_images, test_xmls, test_size=0.5, random_state=42)

def move_files(file_list, target_dir):
    for file_path in file_list:
        shutil.copy(file_path, target_dir)
```

Snippet Code 5.2: Dataset split

5.5.2 Image Data Augmentation

Image data augmentation is a method that uses various transformations to artificially enhance diversity, and increase the size of dataset used for training. It helps in preventing overfitting and making the model to become more resilient. In this study, the following data augmentation techniques were used:

Scaling: this involves resizing the image either by zooming in(scaling up) or zooming out(scaling down).

Rotation: this rotates the image at a specified angle, typically within a certain range.

Translation: this shifts the image left-right along x-axis or up-down along y-axis.

The code snippet given below shows the augmentation technique used for this study. ImageDataGenerator is a class in Keras that generates augmented data during training. The rotation technique will rotate the images randomly by 30^0, the scaling technique will zoom in or out the images by up to 20%, and where as the width-shift and height-shift will enable horizontal and vertical shifts by up to 20%.

```
datagen = ImageDataGenerator(
    rotation_range=30,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)
```

Snippet Code 5.3: Image Data Augmentation

The fill_mode technique will fill in any newly created pixels after transformation.

5.5.3 Normalization

In the context of sign language translation, normalization helps to ensure that the input data, image frames in our case, has a consistent scale. This helps in improving the model's convergence and reducing sensitivity to initialization. In this study a min-max scaling, a scaling of pixel values to a specific range of [0,1], is used.

```
def min_max_normalization(image):
    return image / 255.0
```

Snippet Code 5.4:Image Normalization

5.6 Implementation of The Proposed Models

5.6.1 Feature Extraction with CNN

In the proposed model, the input for the encoder LSTM, and the transformer is a feature vector containing a sequence of features extracted from the image frames. The feature extraction is done using pretrained CNN model ResNet50.

```
feature_model = ResNet50(weights="imagenet", include_top=False, pooling="avg")
```

Snippet Code 5.5: Initializing Feature Extraction with ResNet50

```
def extract_features(base_dir, annotations_file, output_features_file, output_labels_file):
    annotations = load_annotations(annotations_file)
    features = []
    labels = []

    print(f"Processing: {base_dir}")
    for phrase, frames in tqdm(annotations.items(), desc=f"Processing {base_dir}"):
        for frame_entry in frames:
            signer = frame_entry["signer"]
            frame = frame_entry["frame"]
            frame_path = os.path.join(base_dir, phrase, signer, frame)

            if not os.path.exists(frame_path):
                print(f"Frame not found: {frame_path}")
                continue
            img = load_img(frame_path, target_size=(224, 224))
            img_array = img_to_array(img)
            img_array = preprocess_input(np.expand_dims(img_array, axis=0))

            feature_vector = feature_model.predict(img_array, verbose=0)[0]
            features.append(feature_vector)
            labels.append(phrase)
```

Snippet Code 5.6: Defining Feature Extraction from Image Frames

5.6.2 Loading Dataset

The preprocessed Afan Oromo Sign Language features and labels dataset were loaded to test the performance of proposed model and verify its effectiveness.

```
def load_features_and_labels(split_name):
    features = np.load(f"{features_dir}{split_name}_features.npy")
    with open(f"{features_dir}{split_name}_labels.json", "r") as f:
        labels = json.load(f)
    return features, np.array(labels)
```

Snippet Code 5.7: Loading Features and Labels

The train and test datasets, which included both the extracted features and their labels, were also loaded. The labels are the encoded using the LabelEncoder(), which normalizes the labels.

```
train_features, train_labels = load_features_and_labels("train")
val_features, val_labels = load_features_and_labels("val")
test_features, test_labels = load_features_and_labels("test")

label_encoder = LabelEncoder()
train_encoded_labels = label_encoder.fit_transform(train_labels)
val_encoded_labels = label_encoder.transform(val_labels)
test_encoded_labels = label_encoder.transform(test_labels)
```

5.6.3 Setting Train Hyperparameters

Hyperparameters are parameters that are configured prior to training a deep learning model and govern the learning process itself. The train hyperparameters specify the dimension of the input feature vector, number of hidden units, batch size, epochs and dimension of the output vocabulary.

```
latent_dim = 256
embedding_dim = 256
output_dim = 15
epochs = 30
```

Snippet Code 5.8: Defining Fixed Hypeparameters

```
batch_sizes = [16, 32, 64]
learning_rates = [1e-4, 5e-4, 1e-3]
dropout_rates = [0.3, 0.5, 0.7]

hyperparameter_combinations = list(itertools.product(batch_sizes, learning_rates, dropout_rates))
```

Snippet Code 5.9: Defining Hypeparameter Tuning for LSTM Encoder-Decoder

```
{"learning_rate": 0.0005, "batch_size": 16, "dropout_rate": 0.1, "hidden_units": 256},
{"learning_rate": 0.0005, "batch_size": 32, "dropout_rate": 0.1, "hidden_units": 128},
{"learning_rate": 0.0005, "batch_size": 32, "dropout_rate": 0.2, "hidden_units": 256},
{"learning_rate": 0.0001, "batch_size": 32, "dropout_rate": 0.2, "hidden_units": 256},
{"learning_rate": 0.0001, "batch_size": 16, "dropout_rate": 0.2, "hidden_units": 256},
```

Snippet Code 5.10: Defining Hypeparameter Tuning for Transformer

5.6.4 Optimizer and Learning Rate Adjustment

For this study, an adam optimizer with an initial learning rate 0.001 is used.

```
adam = keras.optimizers.Adam(lr = 1e-3)
```

Snippet Code 5.11: Optimizer

5.6.5 Defining Model

The proposed LSTM encoder-decoder model's main components are the encoder LSTM, that takes feature vector as an input and returns the final cell states to be decoded, and the decoder LSTM, which takes tokenized word sequence as input and final states of the encoder as an initial state, and returns the resulting translation. The decoder embeds the tokenized words or glosses using an embedding layer, and uses the final states of the encoder as an initial state.

The following code snippets present building the models for the LSTM encoder-decoder and Transformer.

```
def create_model(learning_rate, dropout_rate):
    encoder_inputs = Input(shape=(train_features.shape[1],), name="encoder_inputs")
    encoder_dense = Dense(latent_dim, activation="relu")(encoder_inputs)
    encoder_norm = BatchNormalization()(encoder_dense)
    encoder_dropout = Dropout(dropout_rate)(encoder_norm)
    encoder_reshaped = Reshape((1, latent_dim))(encoder_dropout)

    encoder_lstm = LSTM(latent_dim, return_state=True, name="encoder_lstm")
    _, state_h, state_c = encoder_lstm(encoder_reshaped)

    decoder_inputs = Input(shape=(1,), name="decoder_inputs")
    decoder_embedding = Dense(embedding_dim, activation="relu")(decoder_inputs)
    decoder_reshaped = Reshape((1, embedding_dim))(decoder_embedding)

    decoder_lstm = LSTM(latent_dim, return_sequences=True, return_state=True, name="decoder_lstm")
    decoder_outputs, _, _ = decoder_lstm(decoder_reshaped, initial_state=[state_h, state_c])

    decoder_dense = Dense(output_dim, activation="softmax", name="decoder_dense")
    final_outputs = decoder_dense(decoder_outputs)
```

Snippet Code 5.12: Defining the Encoder and Decoder LSTMs

```
def build_model(input_dim, hidden_units, dropout_rate, output_dim):
    model = Sequential([
        Input(shape=(input_dim,)),
        Dense(hidden_units, activation="relu"),
        Dropout(dropout_rate),
        Dense(output_dim, activation="softmax")
    ])
    return model
```

Snippet Code 5.13: Defining the Transformer

Finally the model is initialized and compiled as follows using the following code snippet.

```
model = Model([encoder_inputs, decoder_inputs], final_outputs)
model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=learning_rate),
    loss="sparse_categorical_crossentropy",
    metrics=["accuracy"],
)
```

Snippet Code 5.14: Model Initialization and Compiling for LSTM encoder-decoder

```
model = build_model(
    input_dim=train_features.shape[1],
    hidden_units=config["hidden_units"],
    dropout_rate=config["dropout_rate"],
    output_dim=len(np.unique(train_labels))
)

model.compile(
    optimizer=Adam(learning_rate=config["learning_rate"]),
    loss="sparse_categorical_crossentropy",
    metrics=["accuracy"]
```

Snippet Code 5.15: Model Initialization and Compiling for Transformer

5.6.6 Train the Model

To train the model, the training loop is defined, and number of epochs are specified. ModelCheckpoint method is employed to monitor the training process,

```
callbacks = [
    ReduceLROnPlateau(monitor="val_loss", factor=0.5, patience=3, min_lr=1e-6),
    ModelCheckpoint(model_save_path, monitor="val_loss", save_best_only=True, verbose=1),
]
# Train the model
history = model.fit(
    [train_features, train_decoder_inputs],
    train_encoded_labels,
    validation_data=([val_features, val_decoder_inputs], val_encoded_labels),
    batch_size=batch_size,
    epochs=epochs,
    class_weight=class_weights,
    callbacks=callbacks,
)
```

Snippet Code 5.16: Training LSTM Encoder-Decoder Model

```
callbacks = [
    ModelCheckpoint(f"{model_save_dir}/model_{i}.keras", save_best_only=True, monitor="val_loss"),
    ReduceLROnPlateau(factor=0.5, patience=3, min_lr=1e-6)
]
# Train
history = model.fit(
    train_features, train_labels,
    validation_data=(val_features, val_labels),
    epochs=30,
    batch_size=config["batch_size"],
    callbacks=callbacks,
    verbose=1
)
histories[f"Model {i}"] = history.history
```

Snippet Code 5.17: Training the Transformer Model

The statement callbacks=[ReduceLROnPlateau, ModelCheckpoint] is used to set up callbacks in order to save the best model.

5.6.6 Implementation of Evaluation Metrics

Model performance was assessed using the following metrics:

Word Error Rate (WER): This metric quantifies the difference between the machine-generated transcription and the corresponding human-provided reference. It essentially measures how many words in the machine output are incorrect compared to the ground truth.

ROUGE (Recall-Oriented Understudy for Gisting Evaluation): ROUGE evaluates the similarity between machine-generated text and human-written reference text by measuring the overlap of words, phrases, and word sequences. Two frequently used ROUGE variants are described below.

```
# Calculate WER
wer_score = wer(true_labels, predictions)

# Calculate ROUGE scores
scorer = rouge_scorer.RougeScorer(['rouge1', 'rouge2', 'rougeL'], use_stemmer=True)
rouge_scores = [scorer.score(true, pred) for true, pred in zip(true_labels, predictions)]
rouge1_f1 = np.mean([score['rouge1'].fmeasure for score in rouge_scores])
rouge2_f1 = np.mean([score['rouge2'].fmeasure for score in rouge_scores])
rougeL_f1 = np.mean([score['rougeL'].fmeasure for score in rouge_scores])
```

Snippet Code 5.18: Implementation of the Evaluation Metrics

CHAPTER SIX

6. RESULTS AND DISCUSSION

6.1 Overview

This chapter details the results of the proposed solution's implementation. A comparative analysis of these findings is presented using graphical and tabular representations.

6.2 Result Discussion

The training and validation accuracies of the different cases for the Afan Oromo sign language to text translations were observed using both the transformer and LSTM encoder-decoder models. These accuracies are calculated using the Python libraries Tensorflow and Keras.

The training and validation accuracies, as well as losses, were measured for 30 epochs, by tuning combinations of different hyperparameters.

6.2.1 Results of the LSTM Encoder-Decoder Model

For our Afaan Oromoo sign language to text translation purpose, five different cases of LSTM encoder-decoder model were studied. Each of these cases represent a combination of the hyperparameters of the LSTM encoder-decoder model.Table 6.1 represents the five cases and the hypeparameters used during the study.

After training the model with the defined hypeparameters, the accuracies and losses were obserevd. Table 6.2 shows the maximum and minimum training and validation accuracies determined after experimenting with the five cases of the LSTM encoder-decoder model, and Table 6.3 illustrates the maximum and minimum training and validation losses for the Afan Oromo sign language translation.

Table 6.1 Hyperparameter tuning for LSTM encoder-decoder model

Hyperparameter	Case 1	Case 2	Case 3	Case 4	Case 5
Learning Rate	0.0001	0.0001	0.0001	0.0001	0.0001
Batch Size	16	32	64	64	64
Dropout Rate	0.5	0.3	0.3	0.5	0.7

Table 6.2 Accuracy of LSTM encoder-decoder cases

Case	Minimum Training Accuracy		Minimum Validation Accuracy		Maximum Training Accuracy		Maximum Validation Accuracy	
	Epoch	Accuracy	Epoch	Accuracy	Epoch	Accuracy	Epoch	Accuracy
1	1	0.1368	1	0.1628	30	0.7939	23	0.4868
2	1	0.1531	1	0.1783	30	0.8859	24	0.5271
3	1	0.1276	1	0.1473	30	0.9475	19	0.5349
4	1	0.1263	1	0.1085	30	0.8296	25	0.5271
5	1	0.1029	1	0.1550	30	0.6941	29	0.4806

Table 6.3 Loss of LSTM encoder-decoder cases

Case	Minimum Training Loss		Minimum Validation Loss		Maximum Training Loss		Maximum Validation Loss	
	Epoch	Loss	Epoch	Loss	Epoch	Loss	Epoch	Loss
1	28	0.9206	17	1.51013	1	4.7840	1	2.4311
2	29	0.5874	13	1.48997	1	4.8471	1	2.4716
3	30	0.3084	18	1.54455	1	4.8962	1	2.5551
4	30	0.7815	25	1.52181	1	4.9425	1	2.5780
5	30	1.4241	24	1.49281	1	4.9984	1	2.5655

After training the model, the trends in the accuracies and losses were observed. Figures 6.1, 6.3, 6.5, 6.7 and 6.9 illustrate the accuracy of the five LSTM encoder-decoder cases, and figures 6.2, 6.4, 6.6, 6.8 and 6.10 illustrate the loss for theses five cases. Finally the comparative analysis for accuracy and loss of the five cases is presented using figures 6.11 and 6.12.

The first case illustrated in figures 6.1 and 6.2 is an LSTM encoder-decoder model with the hyperparameters, learning rate = 0.0001, batch size = 16 and dropout rate =0.5. The minimum validation accuracy is 16.28% observed at epoch 1, and the

minimum training accuracy, 13.68% is also observed at epoch 1. The maximum training accuracy, 79.39 occurs at epoch 30. While the training loss decreases exponentially, with increasing number of epochs, the validation loss decreases to it's optimal value of 1.51 at epoch 17. The performance of the model has an overall WER = 18.34%, ROUGE-1 = 24.25%, ROUGE-2 = 9.40%, and ROUGE-L = 24.25%.

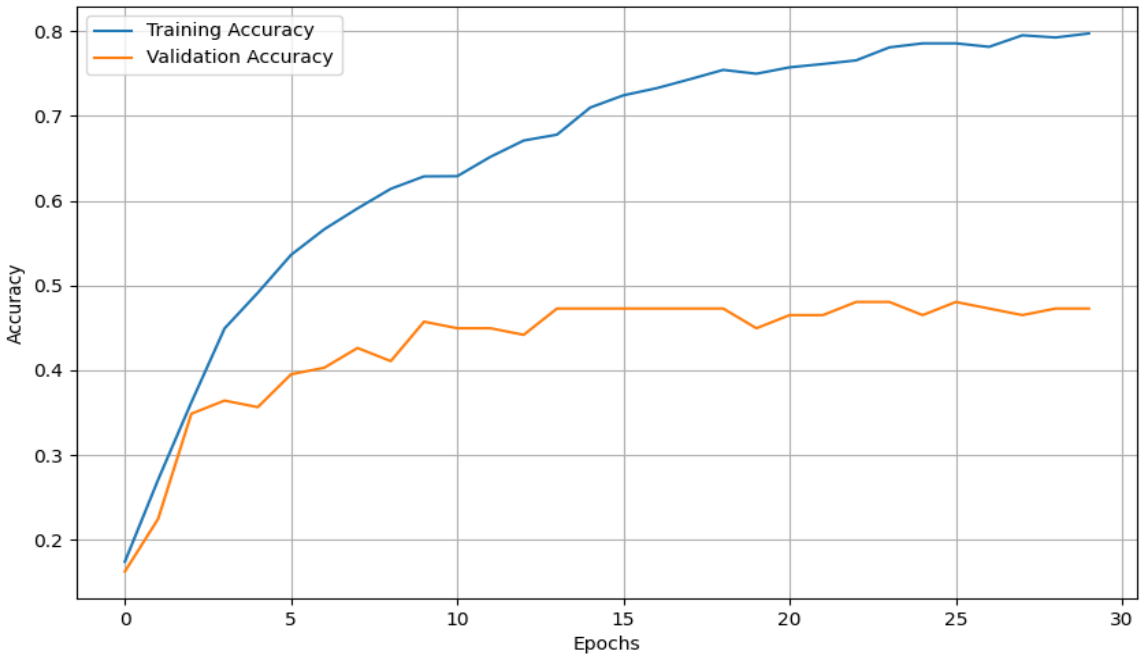


Figure 6.1 Accuracy vs Epoch for Case 1 LSTM

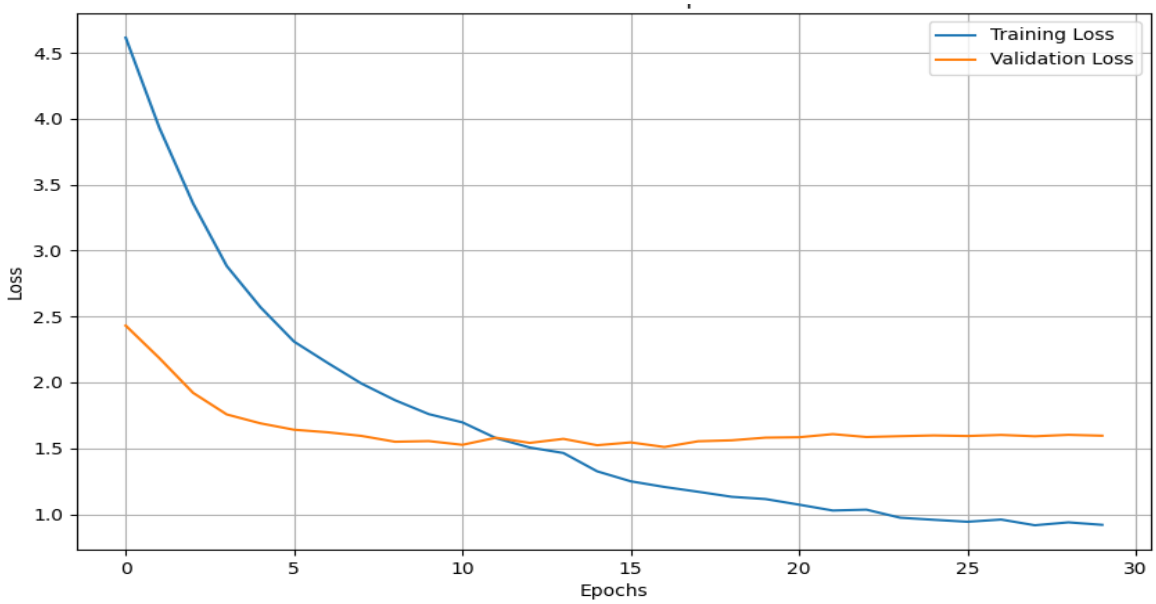


Figure 6.2 Loss vs Epoch for Case 1 LSTM

The second case is an LSTM encoder-decoder model with the hyperparameters, learning rate = 0.0001, batch size = 32 and dropout rate =0.3. The minimum validation accuracy is 17.83% observed at epoch 1, and the minimum training

accuracy, 15.31% is also observed at epoch 1. The maximum training accuracy, 88.59%, occurs at epoch 30. The training loss decreases exponentially from it's peak value of 4.84 to 0.59 at epoch 29, and the validation loss decreases to it's optimal value of 1.49 at epoch 13. The performance of the model has an overall WER = 18.71%, ROUGE-1 = 24.60%, ROUGE-2 = 9.05%, and ROUGE-L = 24.60%. Figures 6.3 and 6.4 present the accuracies and losses of the second case.

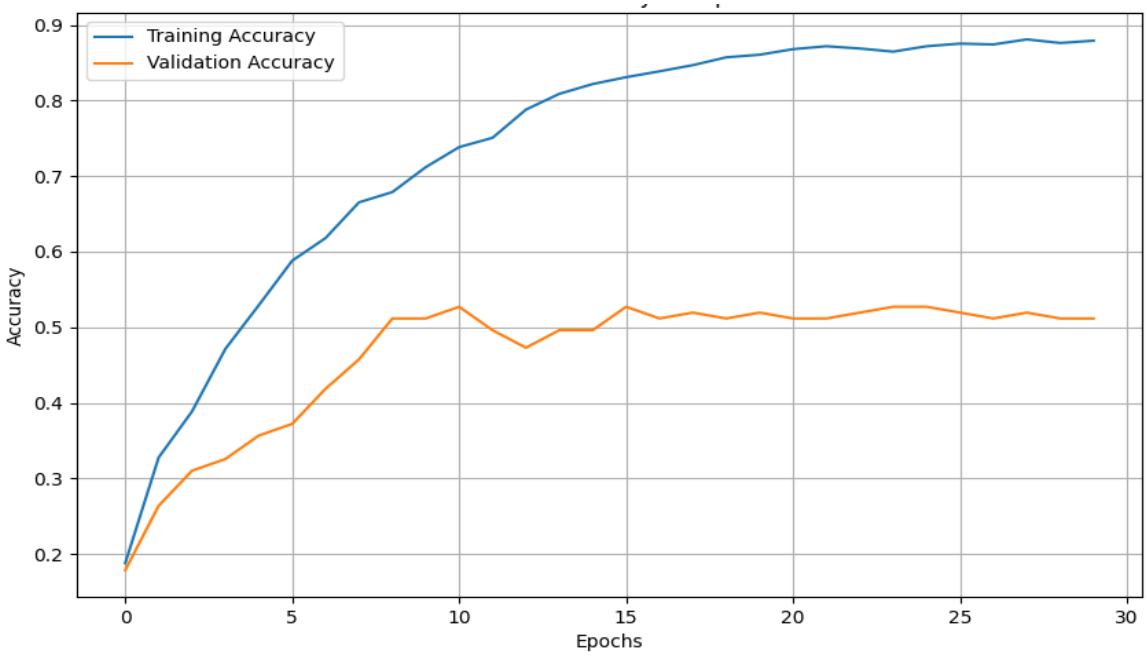


Figure 6.3 Accuracy vs Epoch for Case 2 LSTM

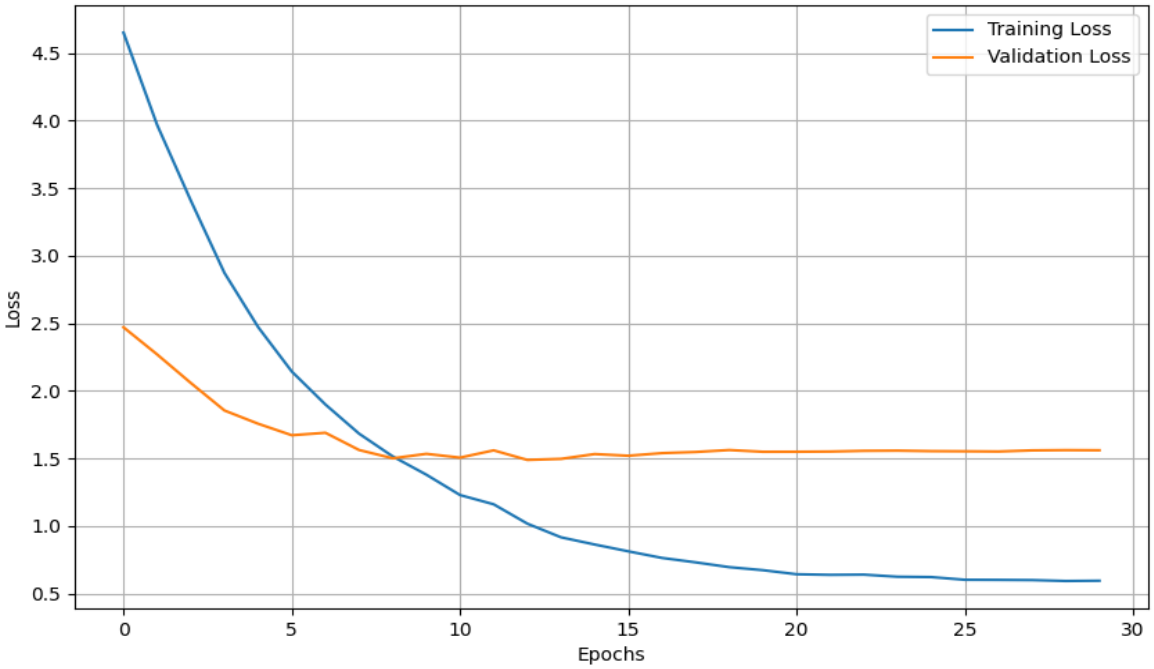


Figure 6.4 Loss vs Epoch for Case 2 LSTM

The next case, illustrated using figures 6.5 and 6.6, is an LSTM encoder-decoder

model with the hyperparameters, learning rate = 0.0001, batch size = 64 and dropout rate =0.3. The minimal training and validation accuracies occur at epoch 1 with the values 12.76% and 14.73% respectively. The maximum training accuracy of 94.75% occurs at epoch 30, and the maximum validation accuracy, 53.49%, occurs at epoch 19. The training loss decreases exponentially, from it's peak value, with the increasing number of epochs. The model achieves it's optimal validation loss of 1.54 at epoch 18. The evaluation for this case yielded WER = 18.34%, ROUGE-1 = 24.25%, ROUGE-2 = 9.40%, and ROUGE-L = 24.25%.

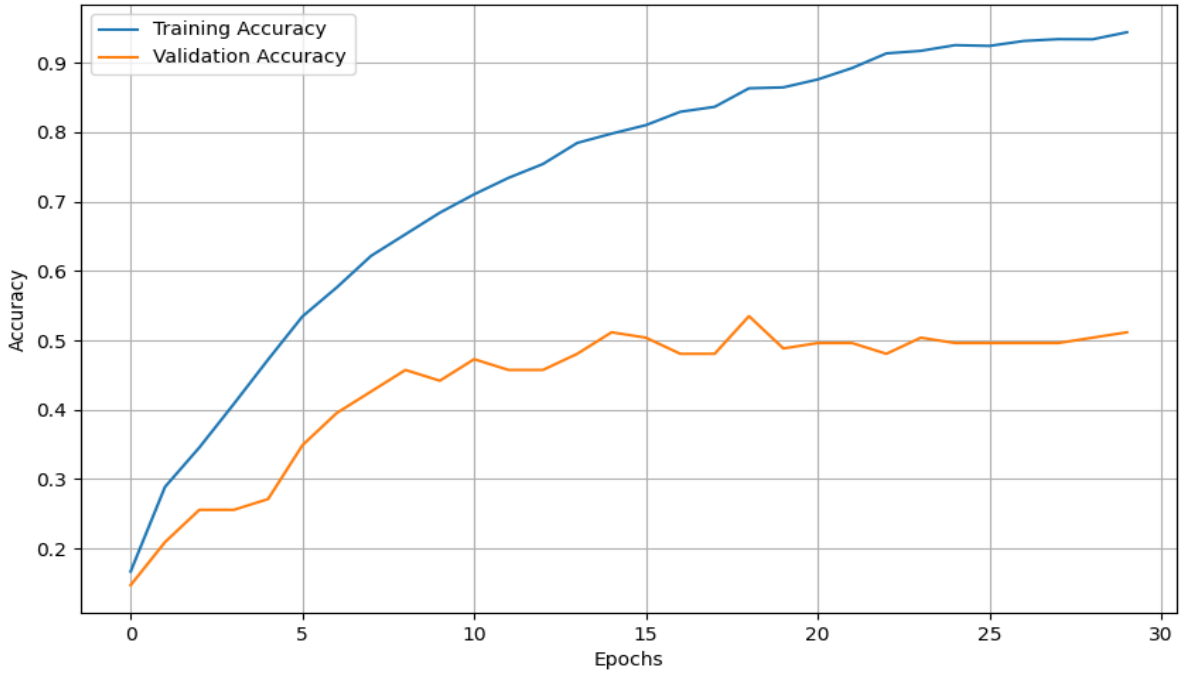


Figure 6.5 Accuracy vs Epoch for Case 3

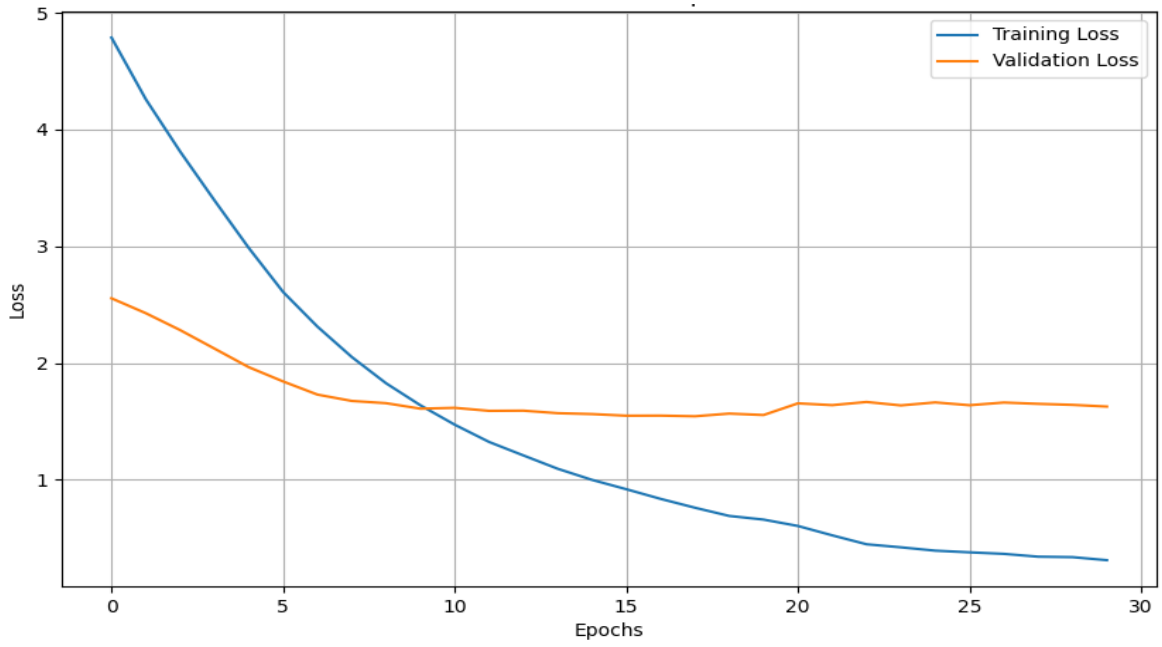


Figure 6.6 Loss vs Epoch for Case 3

The fourth case illustrated in figures 6.7 and 6.8 is an LSTM encoder-decoder model with the hyperparameters, learning rate = 0.0001, batch size = 64 and dropout rate =0.5. The minimum validation accuracy is 10.85% observed at epoch 1, and the minimum training accuracy, 12.63% is also observed at epoch 1. The maximum training accuracy, 82.96%, occurs at epoch 30, and the maximum validation accuracy,52.71%, at epoch 25. While the training loss decreases exponentially to it's optimal value of 0.78 at epoch 30, the validation loss decreases to it's optimal value of 1.52 at epoch 25. The result of the evaluation for this case WER = 18.58%, ROUGE-1 = 23.95%, ROUGE-2 = 9.40%, and ROUGE-L = 23.95%.

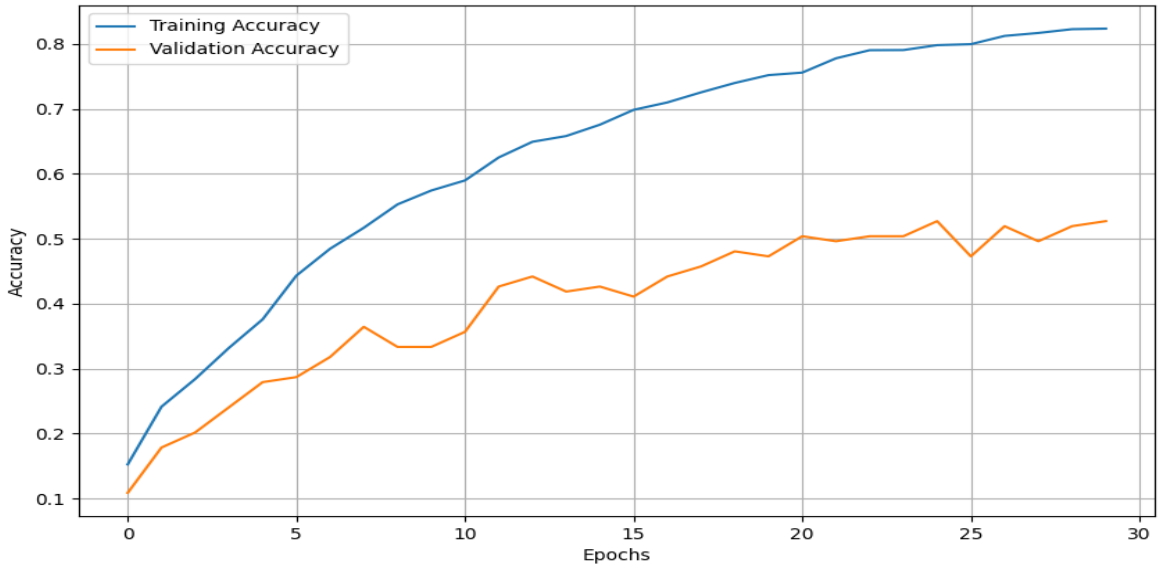


Figure 6.7 Accuracy vs Epoch for Case 4

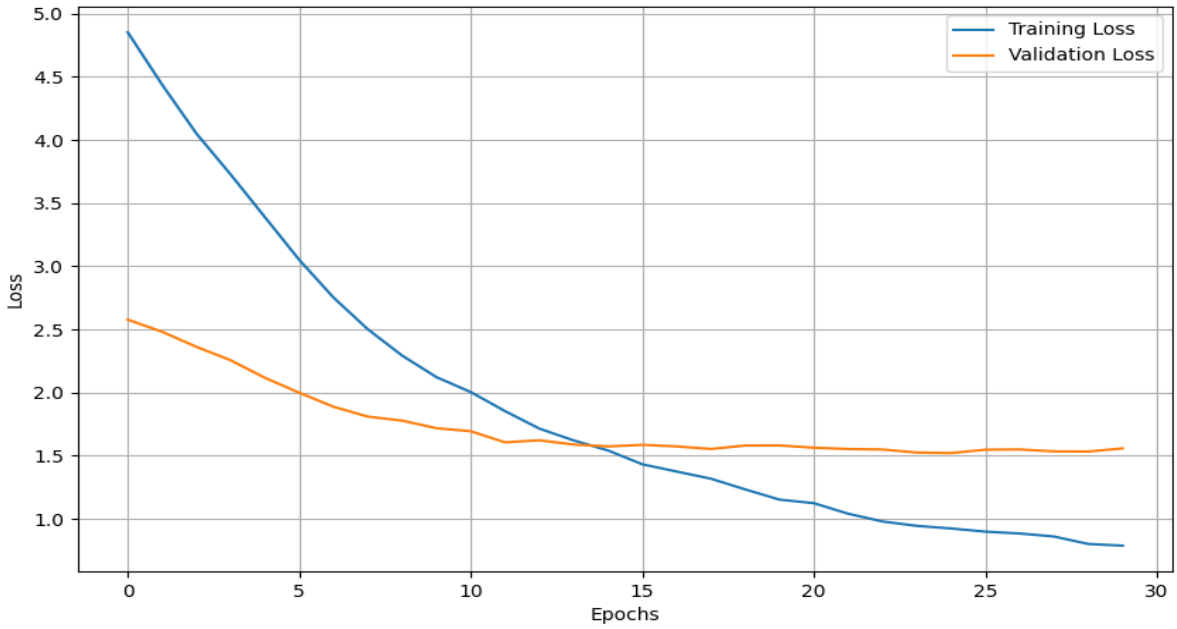


Figure 6.8 Loss vs Epoch for Case 4

The final model with the hyperparameters, learning rate = 0.0001, batch size = 64 and dropout rate =0.7, yielded a minimum validation accuracy of 15.50% at epoch 1, and the minimum training accuracy of 10.29% at epoch 1. The maximum training accuracy, 69.14%, occurs at epoch 30, and the maximum validation accuracy,48.06%, at epoch 29. While the training loss decreases exponentially to it's optimal value of 1.42 at epoch 30, the validation loss decreases to it's optimal value of 1.49 at epoch 24. Figures 6.9 and 6.10 illustrate these results. The results of the evaluation for this case were found to be WER = 18.60%, ROUGE-1 = 23.00%, ROUGE-2 = 9.35%, and ROUGE-L = 23.00%.

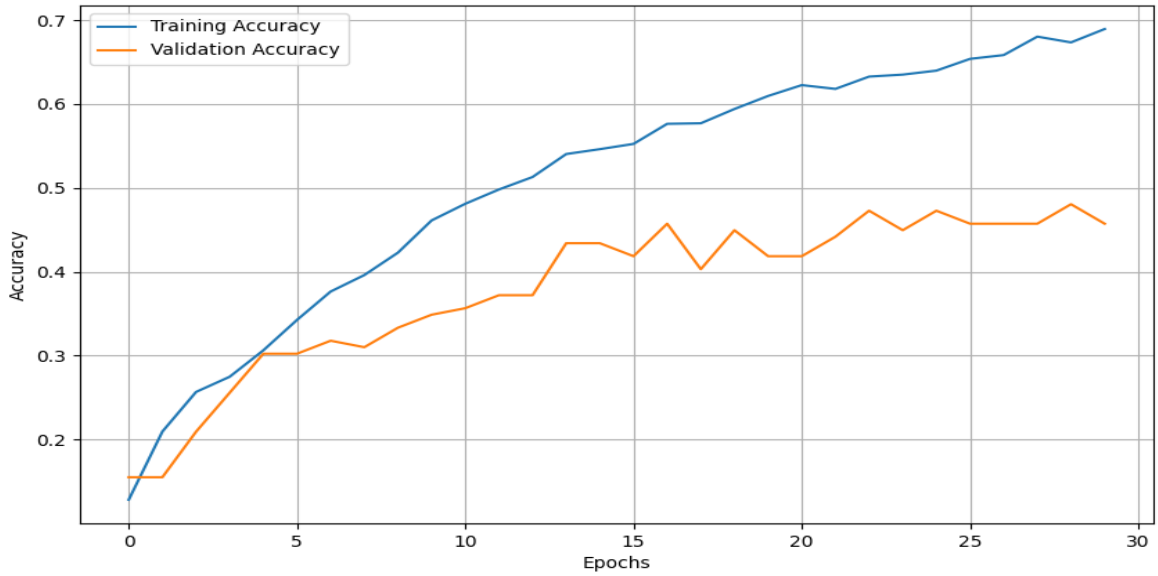


Figure 6.9 Accuracy vs Epoch for Case 5

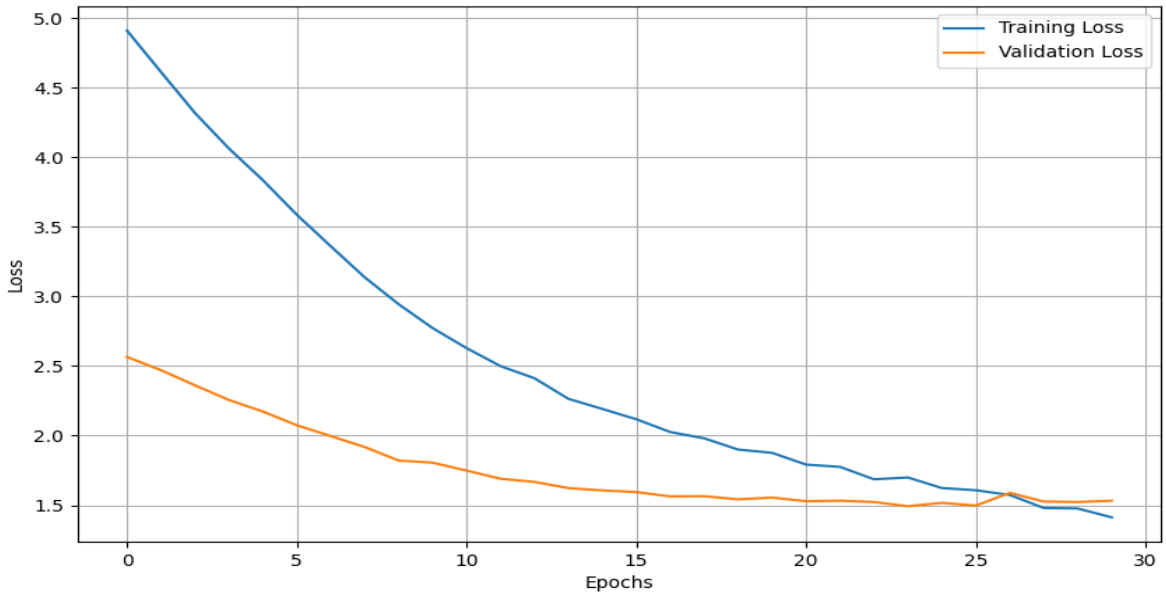


Figure 6.10 Loss vs Epoch for Case 5

The change in performance for Afan Oromo sign language translation was observed using different combinations the hyperparameter values. Figures 6.11 and 6.12 illustrate the comparative analysis for the five LSTM encoder-decoder models based on the evaluation metrics. Accordingly, Case 1 performs slightly better in recognizing the Afan Oromo sign language with the WER of 18.34%. In terms of 1-gram overlap, Case 2 performs better. The 2-gram overlap is nearly the same for all the cases.

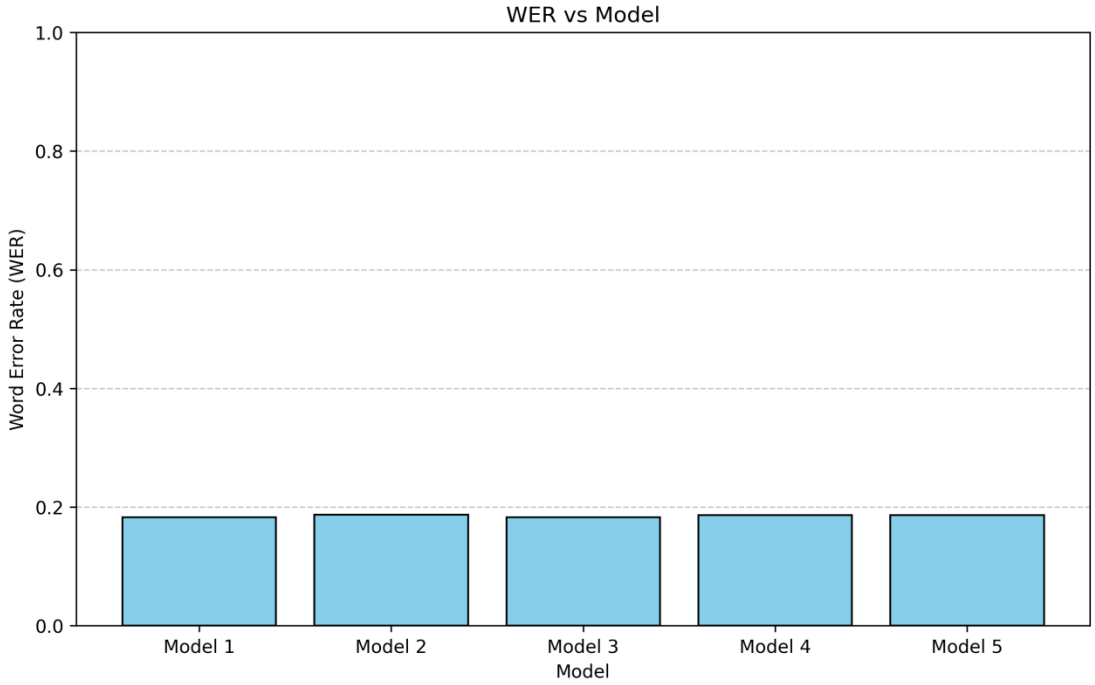


Figure 6.11 WER vs Model for LSTM encodel-decoder

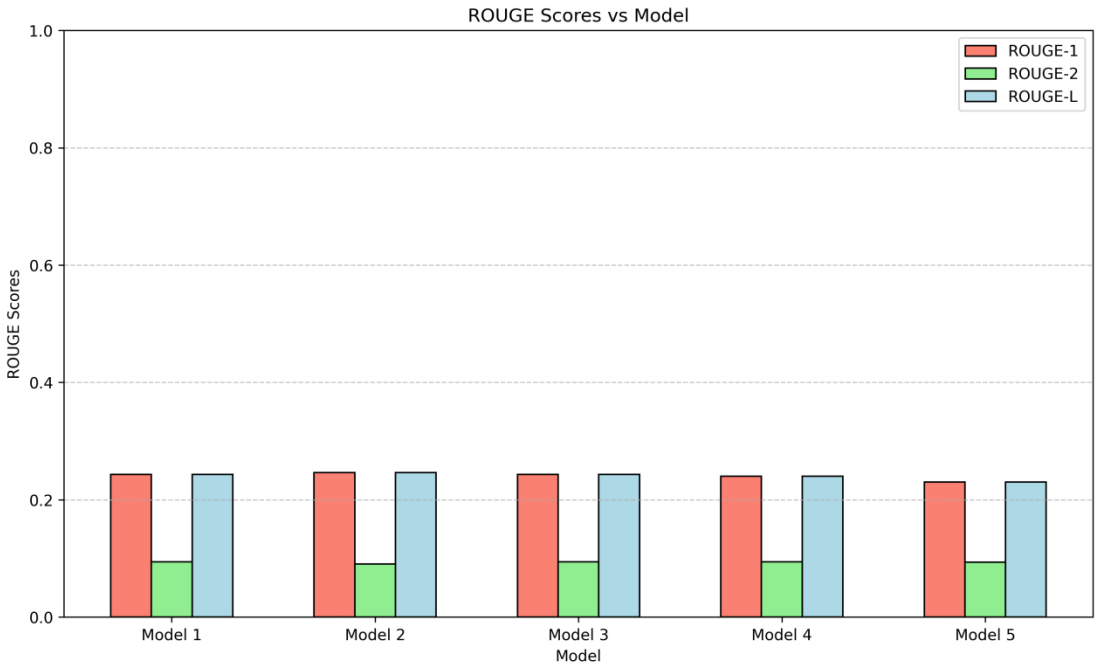


Figure 6.12 ROUGE score vs Model for LSTM encoder-decoder

6.2.2 Results of the Transformer Model

For our Afaan Oromoo sign language to text translation purpose, five different combinations of the transformer hyperparameters we studied. Table 6.4 represents the five cases and the hypeparameter tuning used during the study.

Table 6.4 Hyperparameter tuning for Transformer model

Hyperparameter	Case 1	Case 2	Case 3	Case 4	Case 5
Learning Rate	0.0005	0.0005	0.0005	0.0001	0.0001
Batch Size	16	32	32	32	16
Dropout Rate	0.1	0.1	0.2	0.2	0.2
Hidden Units	256	128	256	256	256

After training the model with the defined hypeparameters, the accuracies and losses were obserevd. Table 6.5 and 6.6 show the maximum and minimum accuracies and losses determined after experimenting with the five cases of the transformer model.

Table 6.5 Accuracy of the Transformer model

Case	Minimum Training Accuracy		Minimum Validation Accuracy		Maximum Training Accuracy		Maximum Validation Accuracy	
	Epoch	Accuracy	Epoch	Accuracy	Epoch	Accuracy	Epoch	Accuracy
1	1	0.2486	1	0.2403	28	0.9738	18	0.5814
2	1	0.2122	1	0.2558	26	0.9336	11	0.5271
3	1	0.2138	1	0.2481	30	0.9399	17	0.5349
4	1	0.1695	1	0.1938	30	0.9347	25	0.5271
5	1	0.1815	1	0.2171	28	0.9387	27	0.5581

Table 6.6 Loss of the Transformer models.

Case	Minimum Training Loss		Minimum Validation Loss		Maximum Training Loss		Maximum Validation Loss	
	Epoch	Loss	Epoch	Loss	Epoch	Loss	Epoch	Loss
1	27	0.1543	13	1.7233	1	2.3294	1	2.0765
2	26	0.2960	11	1.6919	1	2.3992	1	2.2457
3	30	0.2551	9	1.6023	1	2.4342	1	2.2229
4	30	0.3028	26	1.5616	1	2.6265	1	2.3729
5	30	0.2947	14	1.5548	1	2.5647	1	2.3217

The first case illustrated in figures 6.13 and 6.14 is a transformer model with the hyperparameters, learning rate = 0.0005, batch size = 16, dropout rate =0.1 and hidden units = 256. The minimum validation accuracy is 24.03% observed at epoch 1, and the minimum training accuracy, 24.86% is also observed at epoch 1. The maximum training accuracy, 97.38%, occurs at epoch 28. The training loss decreases exponentially, to it's optimal value of 0.1543 at epoch 28, and the validation loss reaches it's optimal value of 1.72 at epoch 13. The performance of the model has an overall WER = 42.17%, ROUGE-1 = 58.95%, ROUGE-2 = 48.15%, and ROUGE-L = 58.95%.

The second case is an transformer model with the hyperparameters, learning rate = 0.0005, batch size = 32, dropout rate = 0.1 and hidden units = 128. The minimum validation accuracy is 25.58% observed at epoch 1, and the minimum training accuracy, 21.22% is also observed at epoch 1. The maximum training accuracy, 93.36%, occurs at epoch 26. The training loss decreases exponentially from it's peak value of 2.399 to 0.296 at epoch 26, and the validation loss decreases to it's optimal value of 1.69 at epoch 11. The performance of the model has an overall WER = 48.49%, ROUGE-1 = 52.78%, ROUGE-2 = 41.98%, and ROUGE-L = 52.78%. Figures 6.15 and 6.16 present the accuracies and losses of the second case.

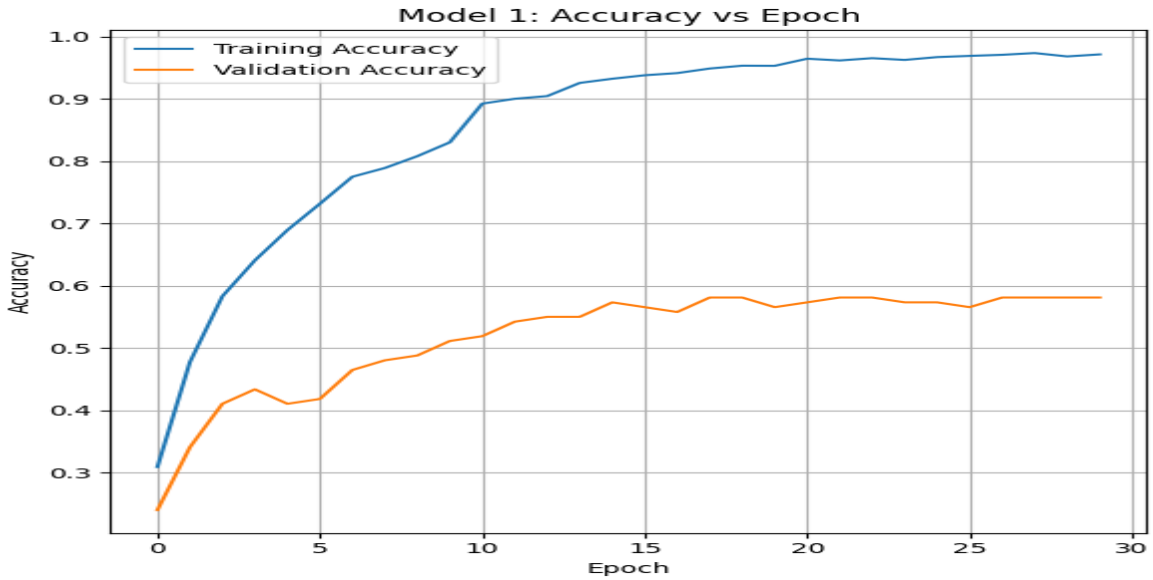


Figure 6.13 Accuracy vs Epoch for case 1

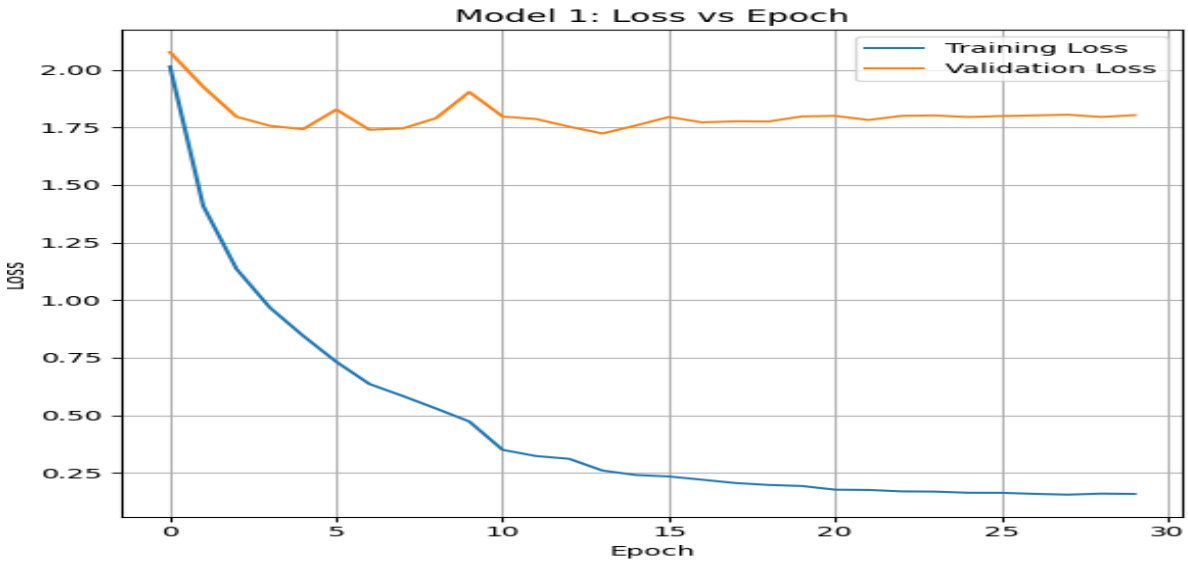


Figure 6.14 Loss vs Epoch for Case 1 Transformer

The next case, illustrated using figures 6.17 and 6.18, is an LSTM encoder-decoder model with the hyperparameters, learning rate = 0.0005, batch size = 32, dropout rate = 0.2 and hidden units = 256. The minimal training and validation accuracies occur at epoch 1 with the values 21.38% and 24.81% respectively. The maximum training accuracy of 93.99% occurs at epoch 30, and the maximum validation accuracy, 53.49%, occurs at epoch 17. The training loss decreases exponentially, from it's peak value to optimal value of 0.255, with the increasing number of epochs. The model achieves it's optimal validation loss of 1.60 at epoch 9 and the validation loss then increases up to epoch 10 before declining for two epochs. The loss then remains constant after epoch 17. The evaluation for this case yielded WER = 46.39%,

ROUGE-1 = 55.25%, ROUGE-2 = 43.21%, and ROUGE-L = 55.25%.

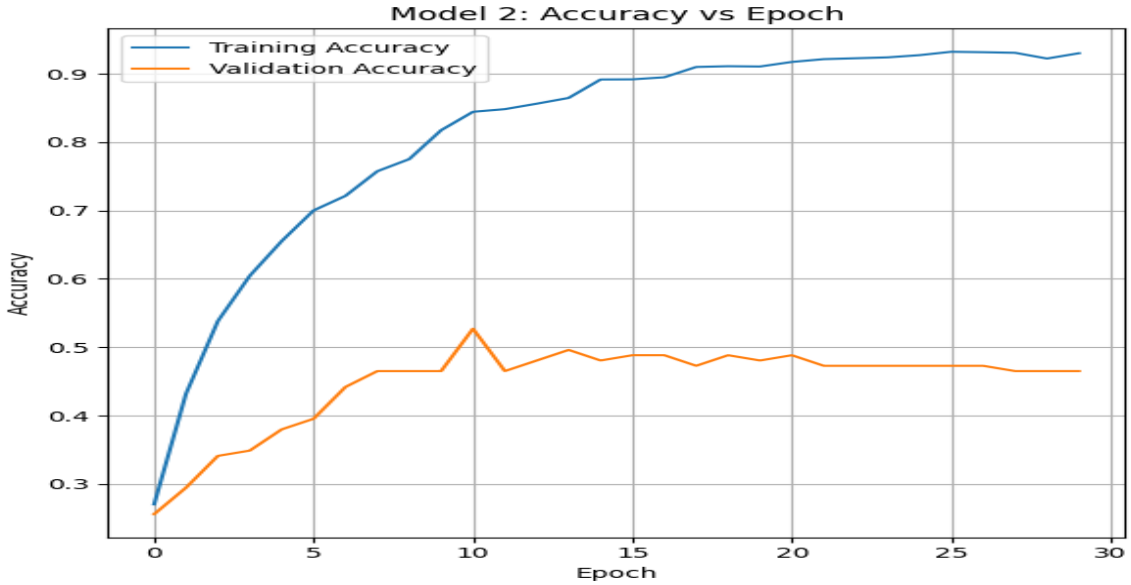


Figure 6.15 Accuracy vs Epoch for Case 2

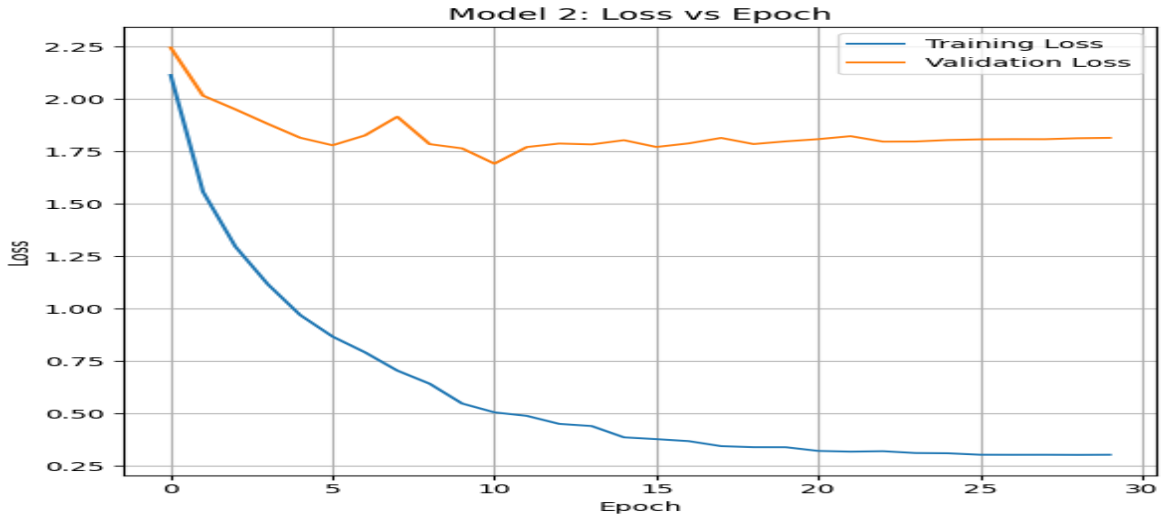


Figure 6. 16 Loss vs Epoch for Case 2

The fourth case illustrated in figures 6.19 and 6.20 is a transformer model with the hyperparameters, learning rate = 0.0001, batch size = 32, dropout rate = 0.2 and hidden units = 256. The minimum validation accuracy is 19.38% observed at epoch 1, and the minimum training accuracy, 16.95% is also observed at epoch 1. The maximum training accuracy, 93.47%, occurs at epoch 30, and the maximum validation accuracy,52.71%, at epoch 25. While the training loss decreases exponentially to it's optimal value of 0.303 at epoch 30, the validation loss decreases to it's optimal value of 1.56 at epoch 26. The result of the evaluation for this case were found to be, WER = 43.98%, ROUGE-1 = 56.17%, ROUGE-2 = 43.83%, and

ROUGE-L = 56.17%.

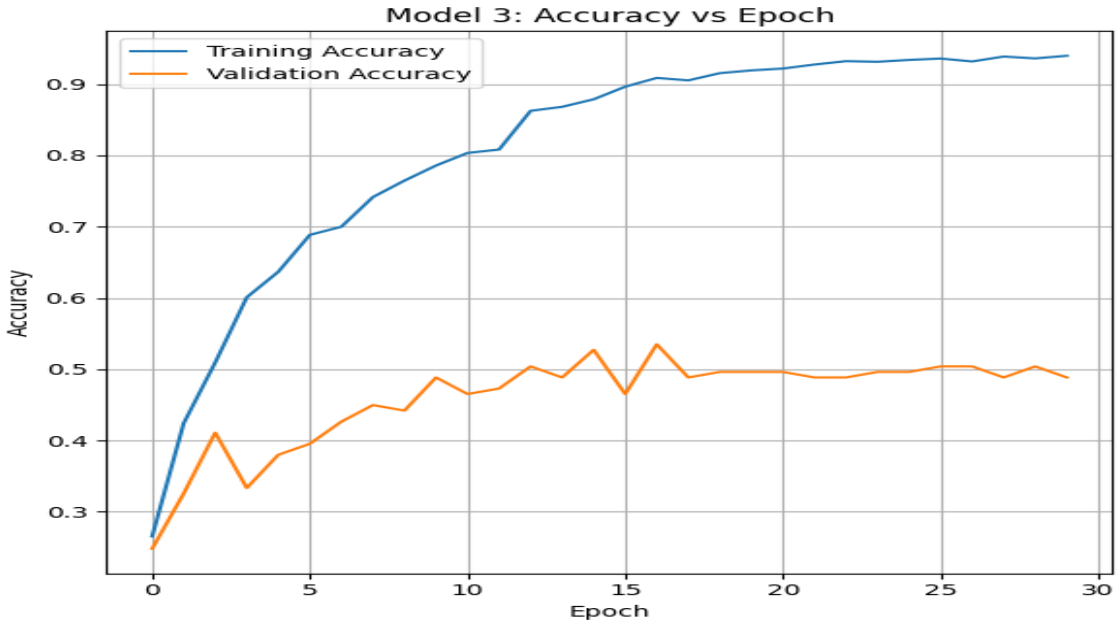


Figure 6.17 Accuracy vs Epoch for Case 3

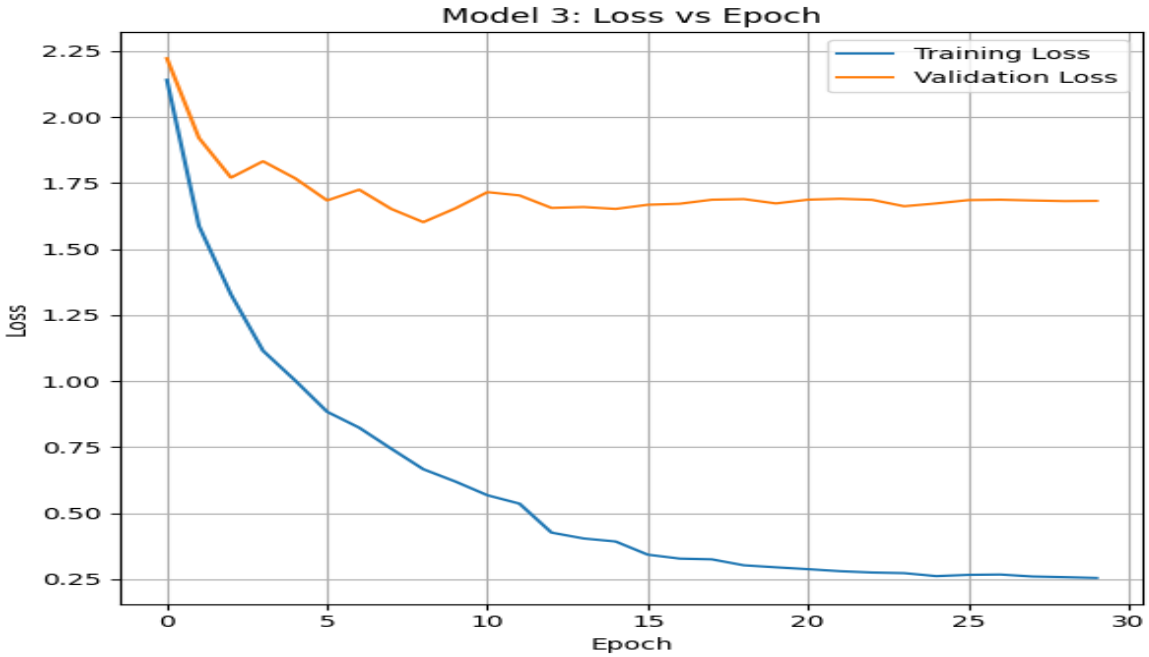


Figure 6. 18 Loss vs Epoch for Case 3

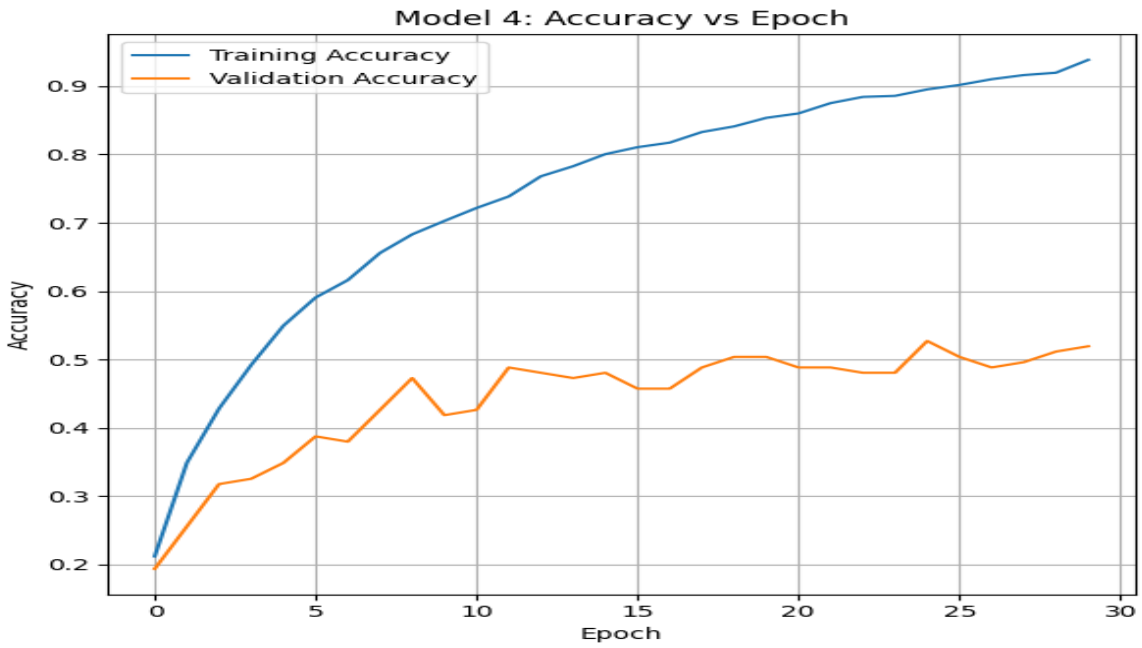


Figure 6. 19 Accuracy vs Epoch for Case 4

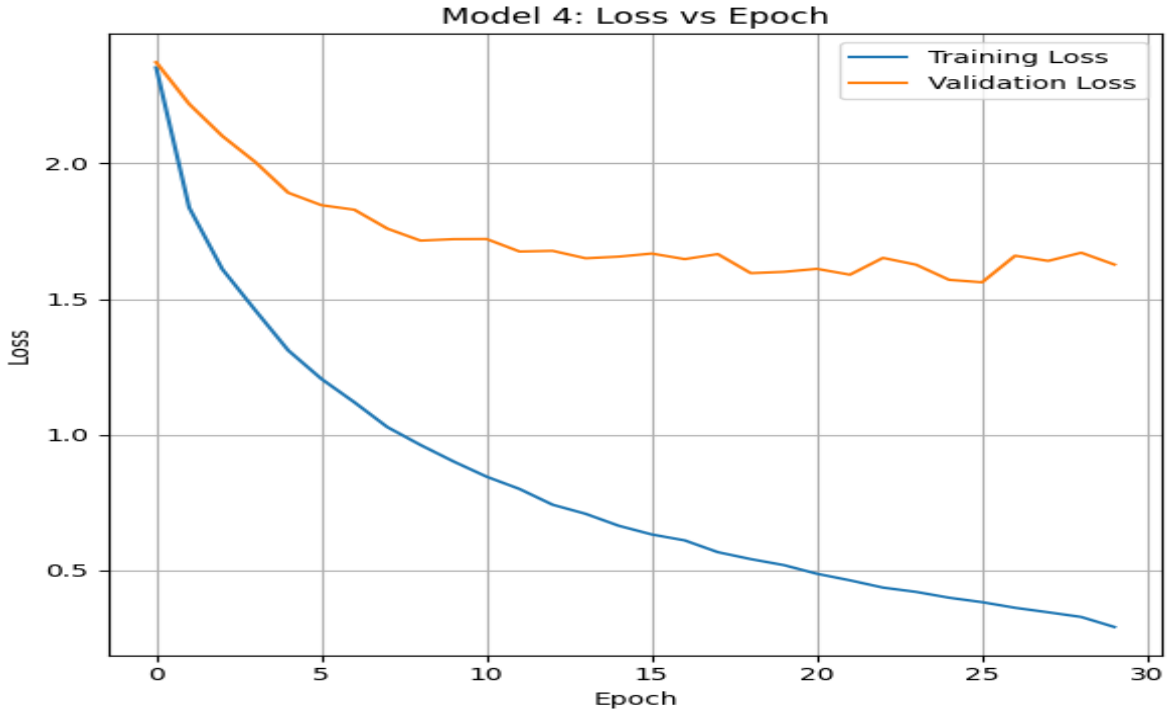


Figure 6.20 Loss vs Epoch for Case 4

The final model with the hyperparameters, learning rate = 0.0001, batch size = 16 dropout rate =0.2 and hidden units = 256, yielded a minimum validation accuracy of 21.71% at epoch 1, and the minimum training accuracy of 18.15% at epoch 1. The1` maximum training accuracy, 93.87%, occurs at epoch 28, and the maximum validation accuracy,55.81%, at epoch 27. While the training loss decreases

exponentially to it's optimal value of 0.295 at epoch 30, the validation loss decreases

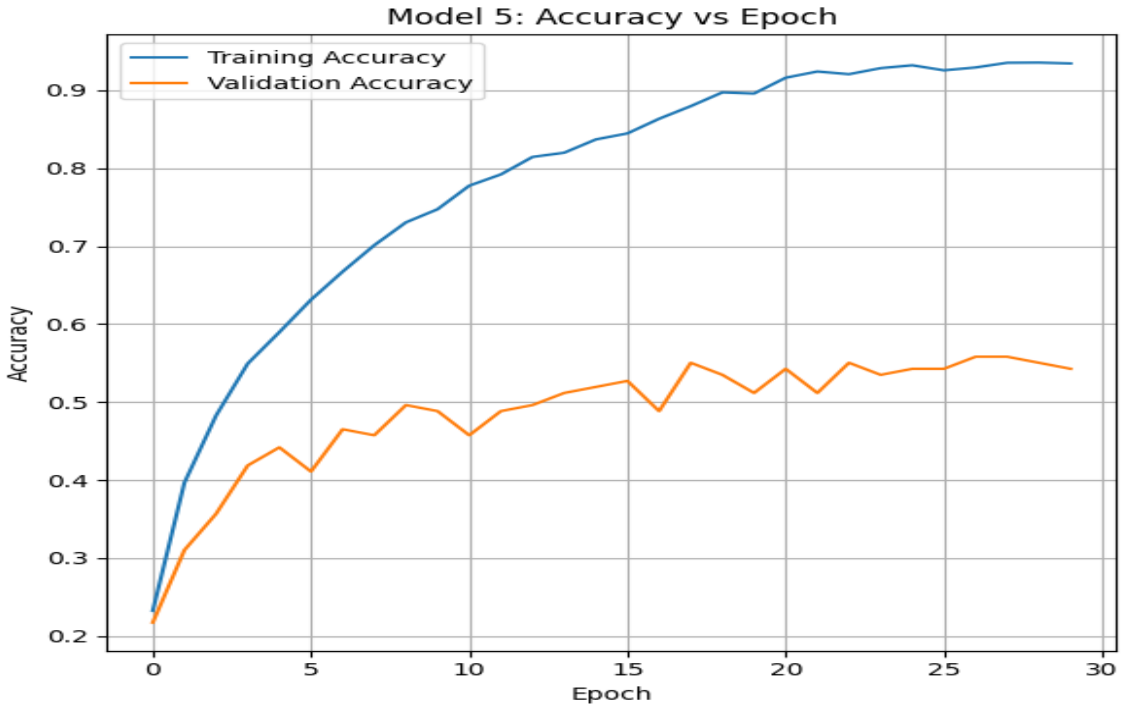


Figure 6.21 Accuracy vs Epoch for Case 5

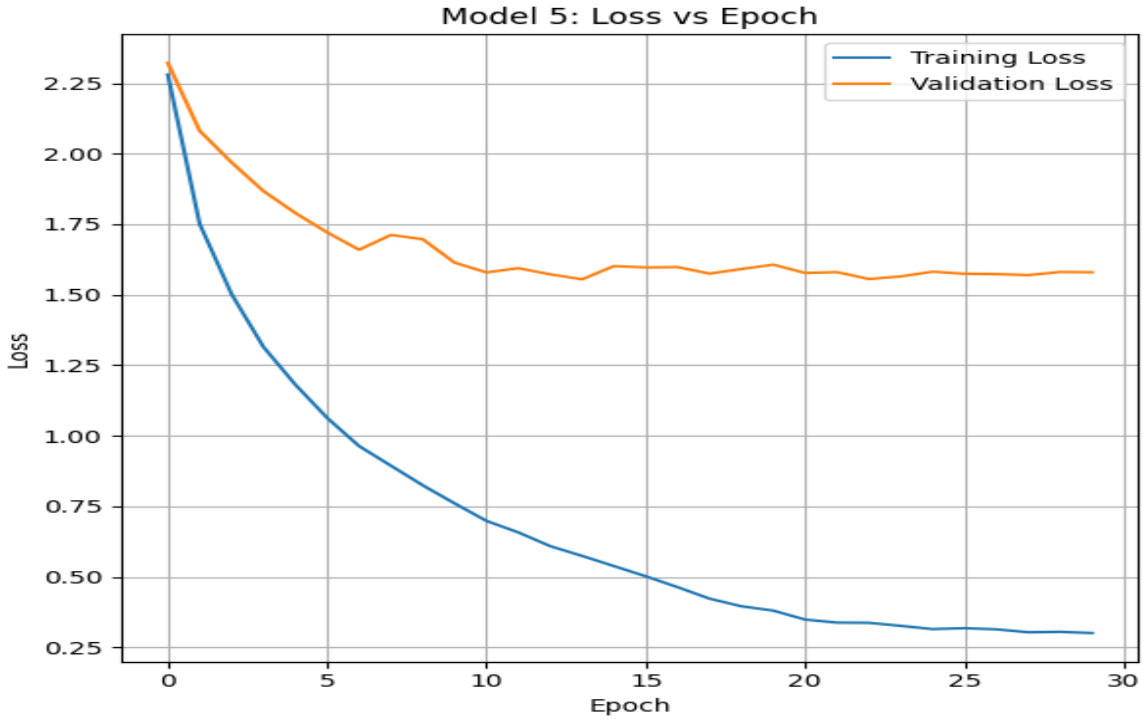


Figure 6.22 Loss vs Epoch for Case 5

The results of the evaluation for this case were found to be WER = 44.58%,ROUGE-1 = 56.17%, ROUGE-2 = 43.83%, and ROUGE-L = 56.17%.

The change in performance for Afan Oromo sign language translation was observed

using different combinations the hyperparameter values. Figures 6.23 and 6.24 illustrate the comparative analysis for the five Transformer models based on the evaluation metrics.

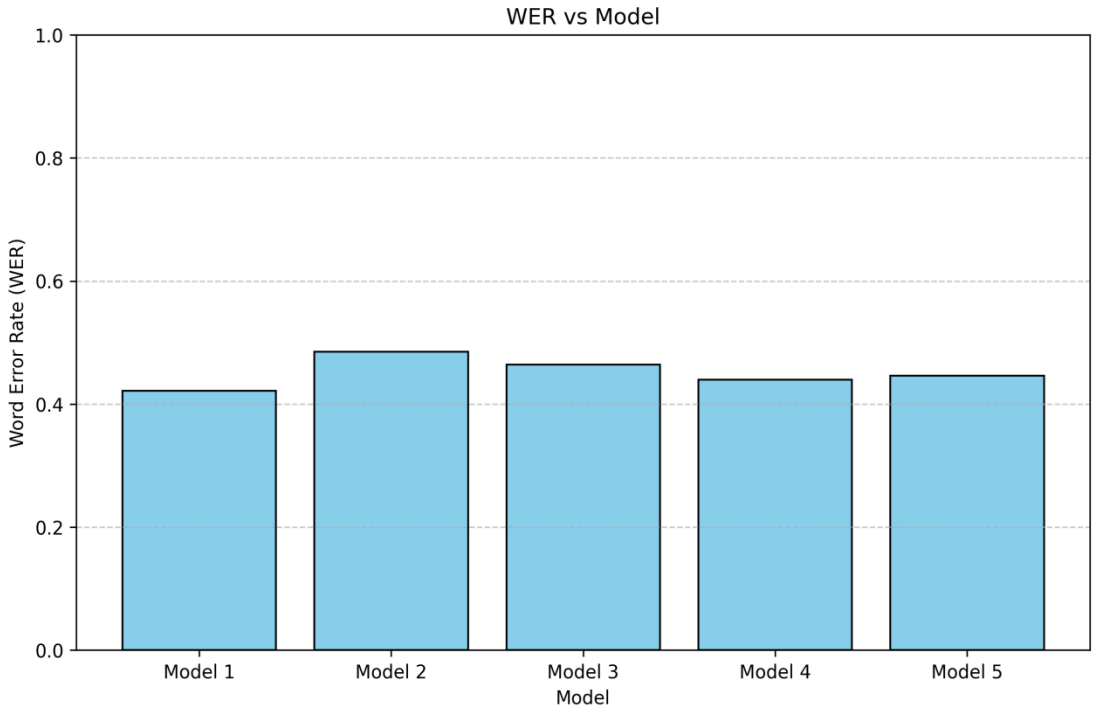


Figure 6.23 WER vs Model for Transformer cases

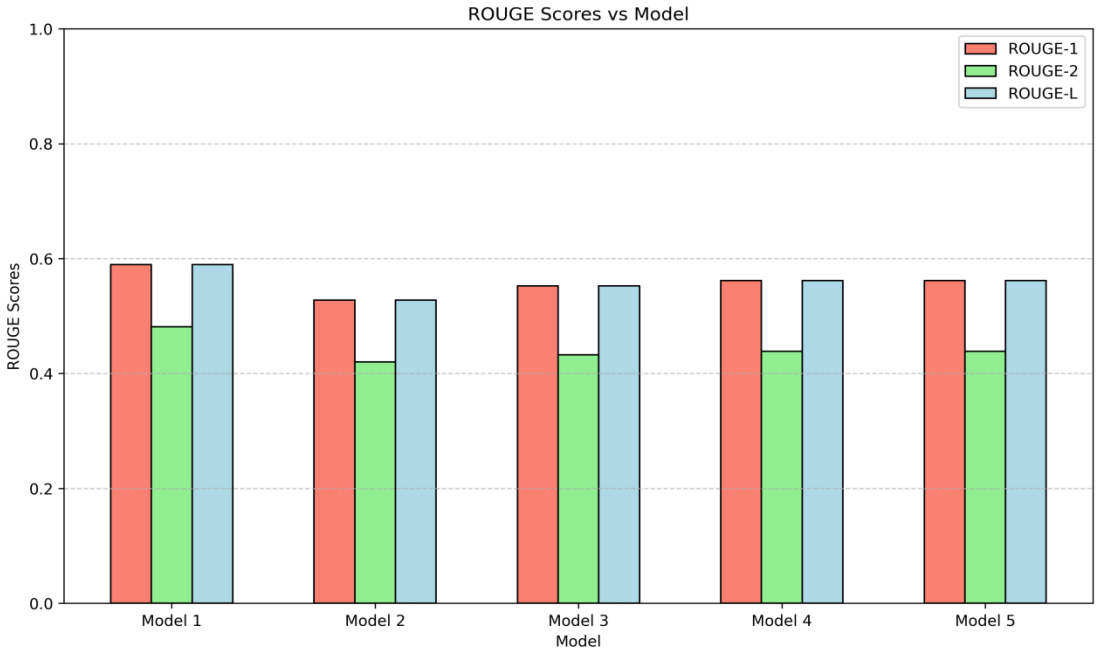


Figure 6.24 ROUGE vs Model for Transformer cases

Accordingly, model 1 performs better in recognizing the Afan Oromo sign language with the WER of 42.17%. In terms of translation, Case 1 has better 1-gram overlap, with ROUGE-score of 58.95, and a 2-gram overlap of 48.15%.

We have conducted the Afan Oromo sign language translation experiments on both LSTM and transformer encoder-decoder models. The comparison between the two models is show using Table 6.7.

Table 6.7 Comparison Of Afan Oromo Sign Language Translation Models

Metric	Best Result from LSTM	Least Result from LSTM	Best Result from Transformer	Least Result from Transformer
WER	18.34%	18.71%	42.17%	48.49%
ROUGE-1	24.60%	23.00%	58.95%	52.78%
ROUGE-2	9.40%	9.05%	48.15	41.98%
ROUGE-L	24.60%	23.00%	58.95%	52.78%

The results show that, the LSTM models have better WER than the transformer models, indicating they performed better in terms of word-level alignment. On the other hand, the transformer models have yielded much better ROUGE scores. This indicates that the transformer models perform better in terms of generating semantically rich translations.

6.3 Research Question Discussion

The research questions raised in the first chapter are discussed below.

RQ1: What deep learning models can be employed for translating Afan Oromo Sign Language into Afan Oromo text?

Answer: In this study, the translation of Afan Oromo sign language into Afan Oromo text is treated as a sequence-to-sequence problem, where the model must effectively capture the gesture sequences and spatial features inherent to sign language. Two deep learning architectures were explored to address this challenge: the LSTM encoder-decoder model and the Transformer encoder-decoder model.

LSTM networks are well-suited for tasks involving sequential data due to their capability to retain information over extended time periods. In our work, the LSTM encoder-decoder framework was used to process sequences of sign language frames and generate corresponding text. The model demonstrated a strong ability to maintain word-level alignment, as reflected in its comparatively lower Word Error Rate (WER). This suggests that LSTMs can effectively model the sequential dependencies in Afan Oromo sign language, even with the constraints of a relatively limited dataset.

However, it was observed that when dealing with longer or more complex sequences, the model occasionally struggled to capture all the subtle details, leading to minor translation errors.

Transformers, on the other hand, use self-attention mechanisms to process entire sequences in parallel, which can result in a better understanding of contextual relationships. In this study, the Transformer model was applied to the same translation task. The self-attention framework allowed the model to generate translations that are semantically richer, as evidenced by higher ROUGE scores. Despite this, the Transformer model exhibited challenges in aligning the input sign language sequences to the correct textual output, resulting in a higher WER. This performance difference is likely due to the larger data requirements of Transformers, which makes them more vulnerable to overfitting.

RQ2: What approaches can be used to enhance the accuracy and efficiency of Afan Oromo Sign Language translation?

Answer: The performance of the Afan Oromo sign language translation techniques can be improved by using diverse dataset. A dataset with a larger number of classes and different signers will have better performance. Other technique used to improve performance of the sign language translation are assigning class-based weights for the sign language phrase classes, and hypeparameter tuning. The hyperparameters tuned for Afan Oromo sign language translation are learning rate, epoch, batch size, dropout rate and number of hidden units.

RQ3: How can dynamic body poses and hand gestures be handled during the sign language recognition and translation?

Answer: Dynamic poses and gestures are important aspects of sign language translation. To handle dynamic poses and gestures, pose estimation algorithm like MediaPipe can be used, during data preprocessing, to extract keypoints for hands and body. Moreover, augmenting the dataset with spatial transformations, like rotations and flips, help in handling dynamic poses and gestures.

CHAPTER SEVEN

7. CONCLUSION AND FUTURE WORKS

7.1 Conclusion

In this study, deep learning models LSTM encoder-decoder and transformer were used to translate Afan Oromo sign language to Afan Oromo text. We recorded fifteen classes of Afan Oromo sign language phrases and sentences, signed by four different signers. We then extracted image frames from the videos, and these frames were preprocessed and annotated. The pre-trained deep convolutional neural network ResNet50 is used to extract features and labels that are used to train and test the models.

We used grid search algorithm to identify the best five hyperparameter tuning configurations for both the LSTM encoder-decoder model and the transformer model. On our dataset, the LSTM encoder-decoder model performed better on the Afan Oromo sign language recognition task, and the transformer model achieved a better result in the Afan Oromo sign language translation task.

Even though much better works have been done on foreign sign languages like American Sign Language(ASL) and German Sign Language translations, only a small number of works have been done in the Afan Oromo Sign Language. Most of these studies focus on Afan Oromo sign language recognition, rather than translation to text. In this research work, we have developed an Afan Oromo sign language translation technique using deep learning technique.

7.2 Future Work

This study has presented Afan Oromo Sign Language to text translation on phrases and sentences. But the longest sequence of words used in this study is a three-word sequence, the sentence “Maqaan Kee Eenyu.” Future works should be able to support phrases and sentences of longer sequence.

We can also see that the dataset is very small is size(1678 frames). We have employed data augmentation to increase the size of our dataset. But even with augmentation, the dataset is small in size, which impacts the accuracy of the models. So future works should increase the size of the dataset and increase the diversity of the signers to get better performance from the models.

Future works should also explore hybrid models that combine the strengths of LSTMs and Transformers, for better translation of sign languages. Additionally, implementing these models in real life settings, and exploring other deep learning models should also be considered in future works.

Special Acknowledgement

This research project is funded by Adama Science and Technology University under the grant number:

ASTU/SM-R/1054/24

Adama, Ethiopia

REFERENCES

Diriba Negash Tesso, Etana Fikadu Dinsa, Hawi Fikadu Kenani. Signed Language Translation into Afaan Oromo Text Using Deep-Learning Approach. American Journal of Artificial Intelligence. Vol. 7, No. 2, 2023, pp. 40-51. doi:10.11648/j.ajai.20230702.12

Muktar Bedaso, Antu Hussein, Chala Diriba. Developing Sign Language Recognition Model for Afaan Oromoo Words Using a Deep Learning Techniques. *Research Square.* 2024; ():.

Mohamed Amin, Hesahm Hefny and Ammar Mohammed, "Sign Language Gloss Translation using Deep Learning Models" International Journal of Advanced Computer Science and Applications(IJACSA), 12(11), 2021. http://dx.doi.org/10.14569/IJACSA.2021.0121178

Farooq, U., Mohd Rahim, M.S. & Abid, A. A multi-stack RNN-based neural machine translation model for English to Pakistan sign language translation. *Neural Comput & Applic* **35**, 13225–13238 (2023). https://doi.org/10.1007/s00521-023-08424-0

Wadhawan, A., Kumar, P. Deep learning-based sign language recognition system for static signs. *Neural Comput & Applic* **32**, 7957–7968 (2020). https://doi.org/10.1007/s00521-019-04691-y

D. Avola, M. Bernardi, L. Cinque, G. L. Foresti and C. Massaroni, "Exploiting Recurrent Neural Networks and Leap Motion Controller for the Recognition of Sign Language and Semaphoric Hand Gestures," in *IEEE Transactions on Multimedia*, vol. 21, no. 1, pp. 234-245, Jan. 2019, doi: 10.1109/TMM.2018.2856094.

Yigzaw, Netsanet & Meshesha, Million & Diriba, Chala. (2022). A Generic Approach towards Amharic Sign Language Recognition. Advances in Human-Computer Interaction. 2022. 1-11. 10.1155/2022/1112169.

Bungeroth, Jan and Hermann Ney. "Statistical Sign Language Translation." (2004).

Necati Cihan Camgoz, Oscar Koller, Simon Hadfield, and Richard Bowden. Sign language transformers: Joint end-toend sign language recognition and translation. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020

Sharma, A., Sharma, N., Saxena, Y., Singh, A. and Sadhya, D., Benchmarking deep neural network approaches for Indian Sign Language recognition. Neural Computing and Applications, vol. 33(12), pp.6685-6696, 2021

Abdul, W., Alsulaiman, M., Amin, S.U., Faisal, M., Muhammad, G., Albogamy, F.R., Bencherif, M.A. and Ghaleb, H., Intelligent real-time Arabic sign language classification using attention-based inception and BiLSTM. Computers & Electrical Engineering, vol. 95, p.107395, 2021

Mohamed Amin, Hesahm Hefny and Ammar Mohammed, "Sign Language Gloss Translation using Deep Learning Models" International Journal of Advanced Computer Science and Applications(IJACSA), 12(11), 2021. http://dx.doi.org/10.14569/IJACSA.2021.0121178

Othman and M. Jemni, "Statistical sign language machine translation: from english written text to american sign language gloss," *arXivpreprint arXiv:1112.0168*, 2011.

Fang Biyi, Co Jillian, and Zhang Mi. 2017. DeepASL: Enabling ubiquitous and non-intrusive word and sentence-level sign language translation. In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*. 1–13

A. Othman and M. Jemni, "Statistical sign language machine translation: from english written text to american sign language gloss," *arXiv preprint arXiv:1112.0168*, 2011.

W. C. Stokoe Jr, "Sign language structure: An outline of the visual communication systems of the american deaf," *Journal of deaf studies and deaf education*, vol. 10, no. 1, pp. 3–37, 2005.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *31st Conference on Neural Information Processing Systems (NIPS 2017)*, 2017.

´Alvarez, Patricia Cabot, Xavier Gir´o Nieto and Laia Tarr´es Benet. "Sign Language Translation based on Transformers for the How2Sign Dataset." (2022).

L. Deng and D. Yu, "Deep learning: methods and applications," *Foundations and trends in signal processing*, vol. 7, no. 3–4, pp. 197– 387, 2014.

Biyi Fang, Jillian Co, and Mi Zhang. 2017. DeepASL: Enabling ubiquitous and non intrusive word and sentence-level sign language translation. In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*. 1–13.

Goodfellow, Ian & Pouget-Abadie, Jean & Mirza, Mehdi & Xu, Bing & Warde-Farley, David & Ozair, Sherjil & Courville, Aaron & Bengio, Y.. (2014).

Generative Adversarial Networks. Advances in Neural Information Processing Systems. 3. 10.1145/3422622.

Necati Cihan Camgoz, Oscar Koller, Simon Hadfeld, and Richard Bowden. 2020. Sign language transformers: Joint end-to-end sign language recognition and translation. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR*. Computer Vision Foundation / IEEE, 10020–10030. https://doi.org/10.1109/CVPR42600.2020.01004

Farooq, U., Rahim, M.S.M., Sabir, N. *et al.* Advances in machine translation for sign language: approaches, limitations, and challenges. *Neural Comput & Applic* **33**, 14357–14399 (2021). https://doi.org/10.1007/s00521-021-06079-3

De Coster, M., Shterionov, D., Van Herreweghe, M. *et al.* Machine translation from signed to spoken languages: state of the art and challenges. *Univ Access Inf Soc* (2023). https://doi.org/10.1007/s10209-023-00992-1

Boulares, M., Jemni, M. Learning sign language machine translation based on elastic net regularization and latent semantic analysis. *Artif Intell Rev* **46**, 145–166 (2016). https://doi.org/10.1007/s10462-016-9460-3

Dhanjal, A.S., Singh, W. An optimized machine translation technique for multi-lingual speech to sign language notation. *Multimed Tools Appl* **81**, 24099–24117 (2022). https://doi.org/10.1007/s11042-022-12763-w

Brour M and Benabbou A 2019 ATLASLang MTS 1: Arabic text language into Arabic sign language machine translation system. Procedia Comput Sci 148(Icds 2018): 236–245. https://doi.org/10.1016/j.procs. 2019.01.066

Cheok, M.J., Omar, Z. & Jaward, M.H. A review of hand gesture and sign language recognition techniques. *Int. J. Mach. Learn. & Cyber.* **10**, 131–153 (2019). https://doi.org/10.1007/s13042-017-0705-5

Ghotkar, A., Barde, U., Sonawane, S. *et al.* Vision-Based Multilingual Sign Language Translation. *SN COMPUT. SCI.* **4**, 715 (2023). https://doi.org/10.1007/s42979-023-02273-3

Fang Biyi, Co Jillian, and Zhang Mi. 2017. DeepASL: Enabling ubiquitous and non-intrusive word and sentence-level sign language translation. In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*. 1–13.

2018. DeepASL: Enabling ubiquitous and non-intrusive word and sentence-level sign language translation. 701 arXiv:1802.07584. Retrieved from http://arxiv.org/abs/1802.07584.

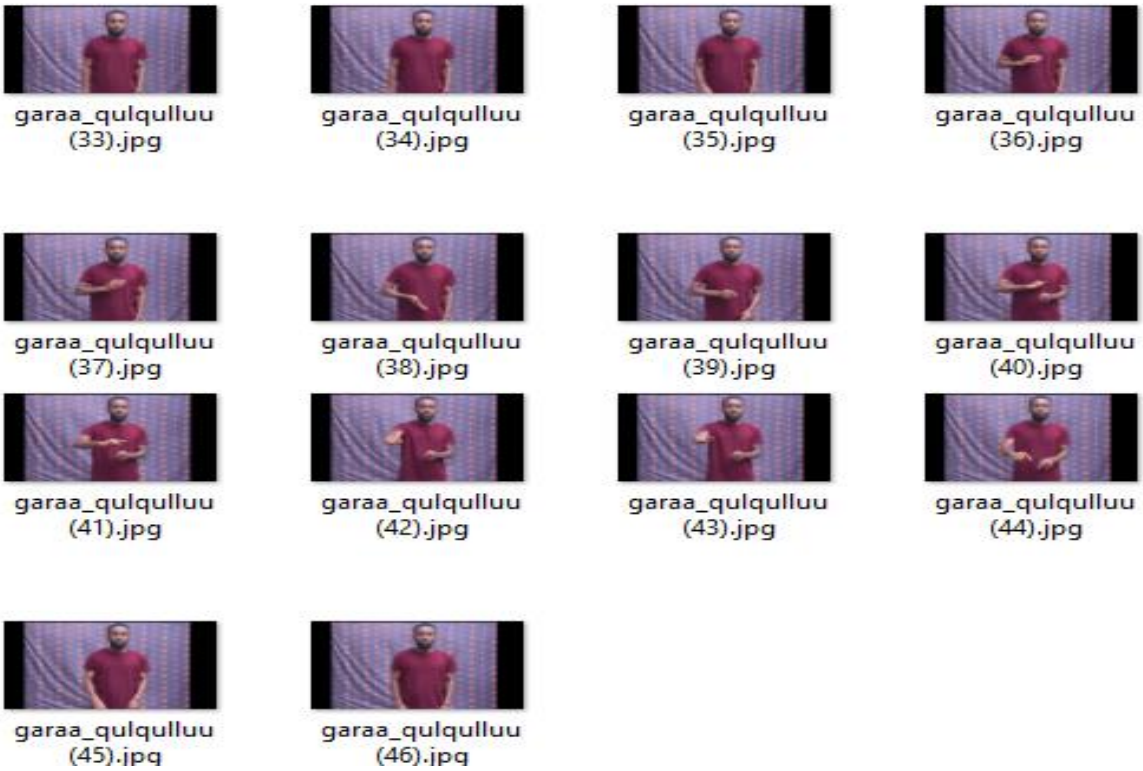
International Day of Sign Languages, United Nations, Accessed 11/28/2022

Eberhard, David M.; Simons, Gary F.; Fennig, Charles D., eds. (2021), "Sign language", *Ethnologue: Languages of the World* (24th ed.), SIL International, retrieved 2021-05-15

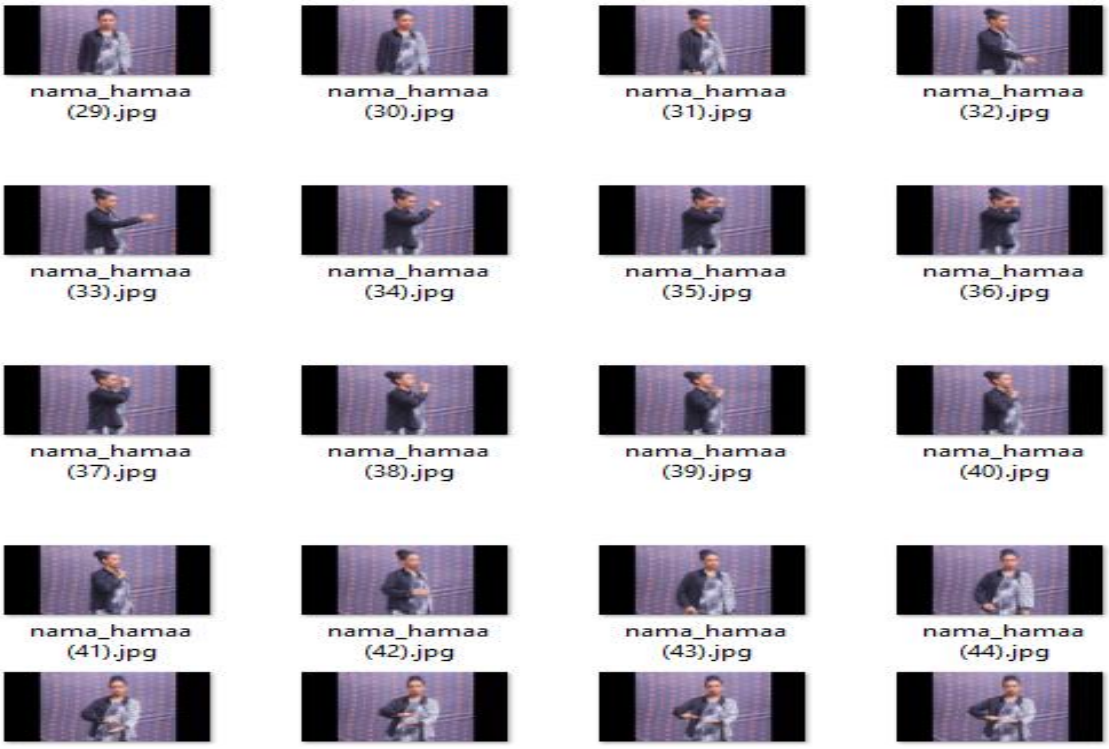
Hosemann, Jana; Steinbach, Markus, eds. (2021), *Atlas of Sign Language Structures*, Sign-hub, archived from the original on 2021-04-13, retrieved 2021-01-13

APPENDICES

Appendix A: Sample Afan Oromo Sign Language Frames



Sample phrase: Garaa Qulqulluu



Sample phrase: Nama Hamaa

Appendix B: Sample Code for The LSTM Encoder-Decoder Model

```
def create_model(learning_rate, dropout_rate):
    encoder_inputs = Input(shape=(train_features.shape[1],), name="encoder_inputs")
    encoder_dense = Dense(latent_dim, activation="relu")(encoder_inputs)
    encoder_norm = BatchNormalization()(encoder_dense)
    encoder_dropout = Dropout(dropout_rate)(encoder_norm)
    encoder_reshaped = Reshape((1, latent_dim))(encoder_dropout)

    encoder_lstm = LSTM(latent_dim, return_state=True, name="encoder_lstm")
    _, state_h, state_c = encoder_lstm(encoder_reshaped)

    decoder_inputs = Input(shape=(1,), name="decoder_inputs")
    decoder_embedding = Dense(embedding_dim, activation="relu")(decoder_inputs)
    decoder_reshaped = Reshape((1, embedding_dim))(decoder_embedding)

    decoder_lstm = LSTM(latent_dim, return_sequences=True, return_state=True, name="decoder_lstm")
    decoder_outputs, _, _ = decoder_lstm(decoder_reshaped, initial_state=[state_h, state_c])

    decoder_dense = Dense(output_dim, activation="softmax", name="decoder_dense")
    final_outputs = decoder_dense(decoder_outputs)

    model = Model([encoder_inputs, decoder_inputs], final_outputs)
    model.compile(
        optimizer=tf.keras.optimizers.Adam(learning_rate=learning_rate),
        loss="sparse_categorical_crossentropy",
        metrics=["accuracy"],
    )
    return model
```

Appendix C: Sample Code for The Transformer Model

```
histories = {}
for i, config in enumerate(top_configs, start=1):
    print(f"\nTraining Model {i} with config: {config}")
    model = build_model(
        input_dim=train_features.shape[1],
        hidden_units=config["hidden_units"],
        dropout_rate=config["dropout_rate"],
        output_dim=len(np.unique(train_labels))
    )

    model.compile(
        optimizer=Adam(learning_rate=config["learning_rate"]),
        loss="sparse_categorical_crossentropy",
        metrics=["accuracy"]
    )

    callbacks = [
        ModelCheckpoint(f"{model_save_dir}/model_{i}.keras", save_best_only=True, monitor="val_loss"),
        ReduceLROnPlateau(factor=0.5, patience=3, min_lr=1e-6)
    ]
  #Training The Model
    history = model.fit(
        train_features, train_labels,
        validation_data=(val_features, val_labels),
        epochs=30,
        batch_size=config["batch_size"],
        callbacks=callbacks
```