A Project Thesis on

CONVERTING SIGN LANGUAGE GESTURES TO SPEECH USING DEEP LEARNING

Submitted in partial fulfilment of the award of degree

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

Submitted by

MORA BALA SAI SATHWICK REDDY [19021A0509]

PEDAMALLU SIREESHA [20025A0561]

KAGITHA BHARGAV [20025A0566]

BEJAWADA MITHILESH CHOWDARY [19021A0508]

Under the supervision of

Dr. S. Chandra Sekhar

Assistant Professor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

UNIVERSITY COLLEGE OF ENGINEERING KAKINADA (A)

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY KAKINADA

KAKINADA - 533003, ANDHRA PRADESH, INDIA

2019-2023

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

UNIVERSITY COLLEGE OF ENGINEERING KAKINADA (A)

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY KAKINADA

KAKINADA - 533003, ANDHRA PRADESH, INDIA

2019-2023

CERTIFICATE

This is to certify that the Project Report entitled **"CONVERTING SIGN LANGUAGE GESTURES TO SPEECH USING DEEP LEARNING"** being submitted by **MORA BALA SAI SATHWICK REDDY** bearing the roll number **19021A0509**, **PEDAMALLU SIREESHA** bearing the roll number **20025A0561**, **KAGITHA BHARGAV** bearing the roll number **20025A0566**, **BEJAWADA MITHILESH CHOWDARY** bearing the roll number **19021A0508** in the partial fulfilment for the award of the degree of **BACHELOR OF TECHNOLOGY** in **COMPUTER SCIENCE AND ENGINEERING** to the **UCEK(A), JNTUK,** Kakinada, Andhra Pradesh, India is a record of bonafide work carried out by them under the guidance and supervision during the academic year 2019-23. It has been found satisfactory and hereby approved for submission.

Signature of Supervisior

Dr. S. Chandra Sekhar

Assistant Professor

Department of CSE

UCEK(A)

JNTUK KAKINADA

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

UNIVERSITY COLLEGE OF ENGINEERING KAKINADA (A)

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY KAKINADA

KAKINADA - 533003, ANDHRA PRADESH, INDIA

2019-2023

CERTIFICATE

This is to certify that the Project Report entitled **"CONVERTING SIGN LANGUAGE GESTURES TO SPEECH USING DEEP LEARNING"** being submitted by **MORA BALA SAI SATHWICK REDDY** bearing the roll number **19021A0509**, **PEDAMALLU SIREESHA** bearing the roll number **20025A0561**, **KAGITHA BHARGAV** bearing the roll number **20025A0566**, **BEJAWADA MITHILESH CHOWDARY** bearing the roll number **19021A0508** in the partial fulfilment for the award of the degree of **BACHELOR OF TECHNOLOGY** in **COMPUTER SCIENCE AND ENGINEERING** to the **UCEK(A), JNTUK,** Kakinada, Andhra Pradesh, India is a record of bonafide work carried out by them under the guidance and supervision during the academic year 2019-23. It has been found satisfactory and hereby approved for submission.

‘

Signature of Head of the Department

Dr. O. Srinivasa Rao

Professor & HOD

Department of CSE

UCEK(A)

JNTUK KAKINADA

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

UNIVERSITY COLLEGE OF ENGINEERING KAKINADA (A)

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY KAKINADA

KAKINADA - 533003, ANDHRA PRADESH, INDIA

2019-2023

DECLARATION

I hereby declare that the work described in this project, entitled **"CONVERTING SIGN LANGUAGE GESTURES TO SPEECH USING DEEP LEARNING"** which is being submitted by our team in partial fulfillment of the requirments for the award of the degree of **BACHELOR OF TECHNOLOGY**, Department of Computer Science and Engineering Kakinada(A), Jawaharlal Nehru Technological University Kakinada, Kakinada- 533003, A.P., is the result of an investigation carried out by our team under the supervision of **Dr. S. Chandra Sekhar**, Assistant Professor, Department of Computer Science and Engineering Kakinada(A), Jawaharlal Nehru Technological University Kakinada, Kakinada- 533003.

The results emboided in this dissertation have not been submitted to any other University or Institute for the award of any degree or diploma.

MORA BALA SAI SATHWICK REDDY [19021A0509]

PEDAMALLU SIREESHA [20025A0561]

KAGITHA BHARGAV [20025A0566]

BEJAWADA MITHILESH CHOWDARY [19021A0508]

ACKNOWLEDGEMENTS

The successful completion of any task is not possible without proper suggestions, guidance, and environment. The combination of these three factors acts as a backbone to my **"CONVERTING SIGN LANGUAGE GESTURES TO SPEECH USING DEEP LEARNING"** project.

I wish to thank first my supervisor **Dr. S. Chandra Sekhar**, Assistant Professor, Department of CSE for accepting and supporting us as a project team. I sincerely convey my gratefulness and heartfelt to my honorable and esteemed project guide for his supervision, guidance, encouragement, and counsel throughout my project. Without his invaluable advice and assistance, it would not have been possible for us to complete this project. His words of encouragement have often inspired me and improved my hopes for completing the project work.

I would like to thank **Dr. O. Srinivas Rao**, Professor, and Head of the computer science and engineering department, UCEK, JNTUK for his fabulous support and useful remarks throughout the project.

I am thankful to **Dr. M.H.M. Krishna Prasad**, Principal, University college of engineering Kakinada (Autonomous), JNTUK for his support and enlightenment during the project.

I am thankful to all the PRC members for their continuous suggestions.

I thank all the teaching and non-teaching staff of the department of computer science and engineering for their support throughout my project work.

MORA BALA SAI SATHWICK REDDY [19021A0509]

PEDAMALLU SIREESHA [20025A0561]

KAGITHA BHARGAV [20025A0566]

BEJAWADA MITHILESH CHOWDARY [19021A0508]

ABSTRACT

This research aims to assist individuals with hearing and speech impairments to convey themselves by breaking down communication barriers they may experience in society. Despite significant advances in the field of Deep Learning-based Sign Language Recognition (SLR), numerous issues still need to be solved. SLR system accuracy can be affected by changes in lighting, hand angle, and signing pace. As a result, despite these disparities, this research aims to develop an accurate and robust SLR system capable of distinguishing a wide range of sign language motions.

The proposed deep learning model uses Gated Recurrent Network (GRU) units to discern patterns in sign language gestures. The SoftMax layer is utilized to output the probability of each word corresponding to the provided gesture, which assists in determining any difficulty the model may have discriminating between words. The trained GRU model achieved 95% accuracy on a subset of 15 words from the Word Level American Sign Language (WLASL) dataset.

This research can potentially have a significant impact since it can help bridge the communication gap between deaf and hard-of-hearing groups and hearing ones. Sign Language Recognition can lead to greater inclusion and communication, ultimately leading to a more inclusive society. People with hearing and speech impairments can gain more autonomy and independence in their daily lives by developing an accurate and comprehensive SLR system that allows them to express themselves more freely and participate more fully in society.

In summary, this research has the potential to significantly improve the lives of persons with hearing and speech impairments, improving their quality of life and enabling more social inclusion.

Contents

List of Figures

CHAPTER - 1

INTRODUCTION

CHAPTER - 1
INTRODUCTION

1 Introduction

1.1 Introduction to Sign Language Recognition

Sign language is a mode of communication used by the deaf and hard of hearing community. Sign languages around the world vary greatly and have their own grammatical rules, vocabulary, and structure. Like any language, sign language is an important means of communication, and for deaf individuals, it is their primary mode of communication. Sign Language Recognition (SLR) is an important field that aims to develop systems that can automatically recognize and interpret sign language gestures, allowing for greater accessibility for the deaf and hard of hearing community.

SLR using Deep Learning is a growing area of research that involves developing models capable of recognizing gestures and translating them into text or speech. Deep Learning models are particularly well-suited to SLR because they can automatically learn complex features and patterns from data. This means that instead of manually designing features, which can be time-consuming and challenging, Deep Learning models can learn the features automatically.

There are many different types of SLR systems, ranging from those that use cameras to capture hand gestures to those that use wearable devices like gloves or bracelets. However, most SLR systems consist of three main components: hand detection, hand tracking, and gesture recognition.

The hand detection component identifies the hand in the video feed or image, and isolates it from the background. This is done using techniques such as skin color segmentation, background subtraction, or machine learning-based object detection methods. Once the hand has been detected, the hand tracking component tracks the movement of the hand over time, allowing the system to recognize the gesture.

The gesture recognition component is the most important part of the SLR system, as it is responsible for identifying the sign language gesture and translating it into text or speech. There are many different types of Deep Learning models that can be used for gesture recognition, including Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and their variants like LSTM or GRU.

CNNs are commonly used for hand gesture recognition tasks because they are able to capture spatial information in the input images. The input to a CNN is a sequence of image frames, which are fed into the network one at a time. The CNN processes each frame and outputs a probability distribution over the set of possible gestures.

RNNs are used for temporal modeling and are well-suited for recognizing gestures that involve motion or changes over time. RNNs can process sequences of input data and maintain an internal memory state that allows them to capture long-term dependencies in the input. This is useful for recognizing sign language gestures, as they often involve a sequence of movements.

In recent years, there have been significant advances in SLR using Deep Learning. Researchers have developed new models that are more accurate and robust, and can recognize a wider range of sign language gestures. However, there are still many challenges to overcome, such as dealing with variations in lighting, hand orientation, and speed of signing.

In conclusion, Sign Language Recognition using Deep Learning is an important and growing field that has the potential to make a significant impact on the lives of deaf and hard of hearing individuals. With the continued development of new Deep Learning models and improvements in SLR systems, we can hope to see greater accessibility for the deaf and hard of hearing community in the near future.

1.2 Deep Learning

Deep Learning is a subset of machine learning that involves training neural networks to recognize patterns in data. It is inspired by the structure and function of the human brain and is capable of learning from large and complex datasets. Deep Learning has been used to achieve state-of-the-art results in various applications such as computer vision, natural language processing, speech recognition, and robotics.

Deep Learning models typically consist of multiple layers of artificial neurons, each of which performs a simple computation on its inputs. By stacking these layers together, the model can learn complex representations of the input data. The training process involves adjusting the weights and biases of the neurons to minimize the error between the predicted output and the true output.

One of the key advantages of Deep Learning is its ability to automatically learn features from raw data, without the need for manual feature engineering. This has led to significant improvements in the accuracy and robustness of machine learning models. However, Deep Learning also requires large amounts of data and computing resources, making it computationally expensive and time-consuming to train.

1.2.1 Why Deep Learning?

Deep Learning (DL) models are well-suited for Sign Language Recognition (SLR) because they can handle the complex and dynamic nature of sign language gestures. DL models, such as Recurrent Neural Networks (RNNs), Convolutional Neural Networks (CNNs), and Long Short-Term Memory (LSTM) networks, can learn the underlying patterns in sign language gestures by processing large amounts of data.

One of the main advantages of DL models is their ability to learn features directly from raw data, without the need for manual feature engineering. This is particularly useful for SLR, as sign language gestures involve complex spatial and temporal patterns that are difficult to represent manually. DL models can learn these patterns automatically, making them more accurate and robust than traditional machine learning models.

DL models also have excellent scalability, meaning that they can be trained on large datasets and can handle real-time applications. As the amount of sign language data continues to grow, DL models can be easily adapted to handle the increased volume of data. Furthermore, DL models can be deployed on various platforms, including mobile devices and web browsers, making them accessible to a wider range of users.

1.2.2 When to Apply Deep Learning?

Deep Learning is a powerful tool for solving complex problems, but it is not always the best approach. Here are some scenarios where Deep Learning is well-suited:

1. **Large and complex datasets:** Deep Learning performs best when there is a large and

complex dataset available for training the model.

2. **High-dimensional data:** Deep Learning is effective at handling high-dimensional data such as images, videos, and speech signals.

3. **Pattern recognition:** Deep Learning is excellent at recognizing complex patterns in data, making it well-suited for tasks such as object detection, image segmentation, and speech recognition.

4. **Natural language processing:** Deep Learning has revolutionized natural language processing tasks such as machine translation, sentiment analysis, and text generation.

5. **Unstructured data:** Deep Learning can handle unstructured data such as text, images, and videos, without the need for manual feature engineering.

6. **Time-series data:** Deep Learning is effective at modeling time-series data, such as stock prices, weather forecasts, and medical records.

7. **Reinforcement learning:** Deep Learning has shown great success in reinforcement learning tasks such as playing games and controlling robots.

8. **Recommendation systems:** Deep Learning can be used to build personalized recommendation systems, such as movie or product recommendations.

9. **Generative models:** Deep Learning can be used to generate new data, such as images, music, and text.

10. **Transfer learning:** Deep Learning models can be fine-tuned for a specific task using transfer learning, which is useful when there is limited data available for training.

1.3 Learning Paradigms

1. Supervised Learning
2. Unsupervised Learning
3. Reinforcement Learning

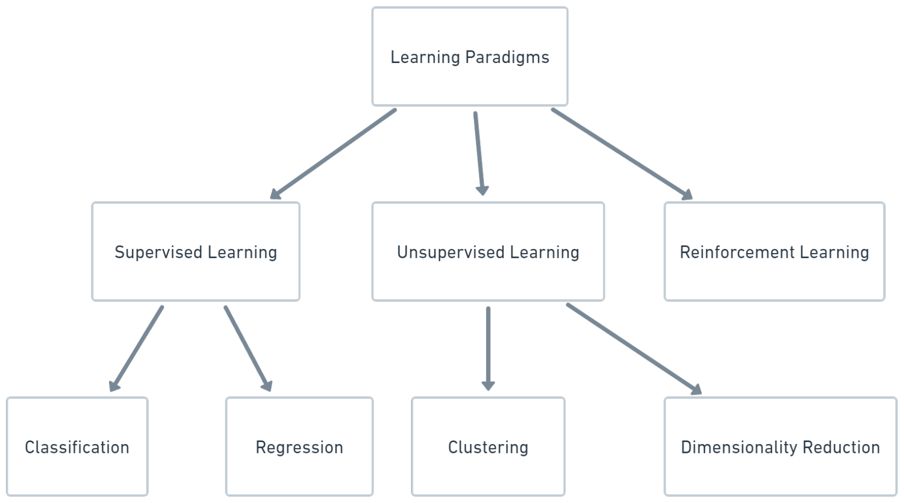


Fig 1: Learning Paradigms in Machine Learning

The three types of learning paradigms, along with the tasks in which they are used, are depicted in Fig 1.

1.3.1 Supervised Learning

Supervised learning is a paradigm where a model is trained using labeled data. Labeled data refers to input data that has corresponding output data, which serves as the ”ground truth” for the model. The model learns to predict the output from the input data by minimizing the difference between its predicted output and the ground truth.

Supervised learning is used to solve a wide range of problems, including classification and regression tasks. In classification tasks, the model learns to classify input data into predefined categories, while in regression tasks, the model learns to predict a continuous output variable.

Supervised learning is effective when there is a large amount of labeled data available for training the model. It is widely used in various applications such as image recognition, natural language processing, and speech recognition.

1.3.2 Unsupervised Learning

Unsupervised Learning is a paradigm where the algorithm is trained on unlabeled data without any specific output labels or targets. Instead, the algorithm is designed to identify patterns or structure within the data on its own. The goal of unsupervised learning is to explore and discover hidden relationships within the data and to group similar data points together. Common unsupervised learning techniques include clustering, dimensionality reduction, and anomaly detection. Unsupervised learning is useful in cases where labeled data is scarce or expensive to obtain, and can provide valuable insights into the underlying structure of the data.

1.3.3 Reinforcement Learning

Reinforcement Learning (RL) is a paradigm where an agent learns to make decisions by interacting with an environment. The agent takes actions in the environment, and the environment provides feedback in the form of rewards or penalties based on the actions taken. The goal of the agent is to learn a policy that maximizes the cumulative reward over time.

In RL, the agent uses trial-and-error to learn the optimal policy. It starts with a random policy and updates it based on the rewards obtained from the environment. The update process involves estimating the value of each state-action pair and using that estimate to improve the policy.

RL has been successful in a variety of applications, including game playing, robotics, and autonomous vehicles. RL algorithms, such as Q-Learning, SARSA, and Deep Q-Networks, have achieved state-of-the-art results in several domains. However, RL can be challenging to implement and requires careful tuning of the reward function and exploration strategy.

1.4 Workflow

A Deep Learning workflow, which is shown in Fig 2, requires careful consideration of data collection, preprocessing, model selection, training, evaluation, and deployment to build a robust and effective model.

1.4.1 Data Collection

Data collection is the process of gathering relevant data from various sources for analysis and decision-making. In the context of machine learning, it involves collecting a large and representative dataset that is used to train the model. The data collected should be diverse, well-annotated, and balanced to ensure the model can generalize well to new data. Data collection

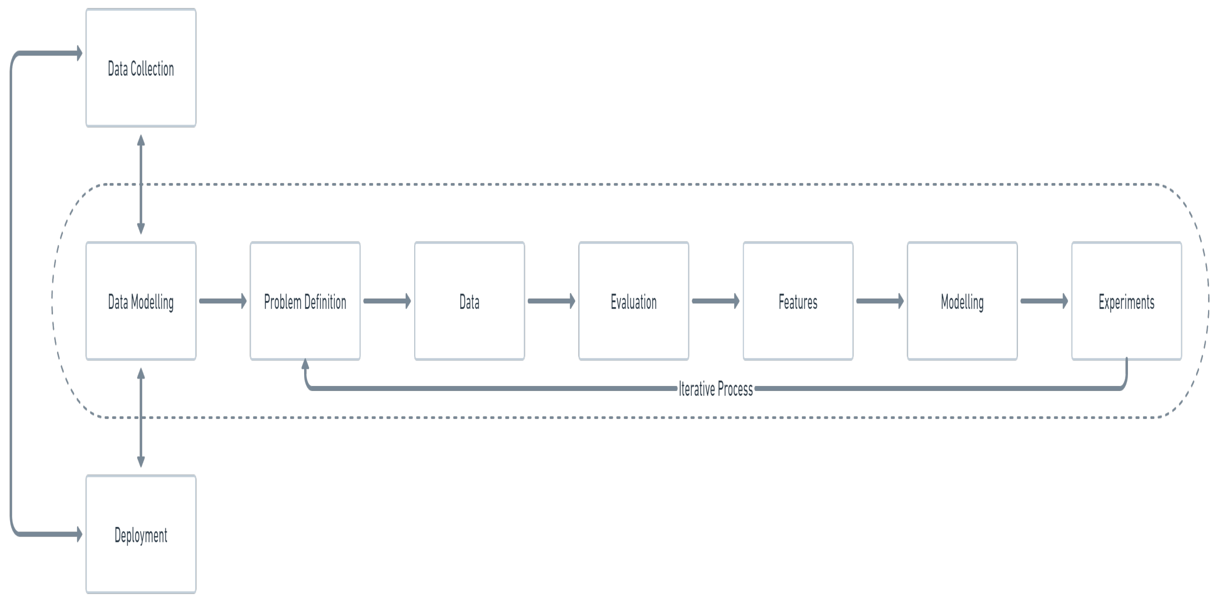


Fig 2: End-to-End Pipeline of Deep Learning

may involve gathering data from public sources, scraping data from websites, or collecting data using sensors or other hardware devices. It is an essential step in the machine learning pipeline and has a significant impact on the performance of the trained model.

1.4.2 Data Preprocessing

Data Preprocessing is a critical step in the Machine Learning workflow that involves preparing the raw data for analysis. It typically involves tasks such as cleaning, normalization, transformation, and feature extraction to ensure that the data is in a format that is suitable for training a model. Data Preprocessing is essential because raw data often contains noise, missing values, and inconsistencies that can lead to inaccurate models. By preprocessing the data, we can improve the quality and reliability of the results obtained from the model. Ultimately, good Data Preprocessing practices can help to ensure that a Machine Learning model is accurate, reliable, and scalable.

The process of Data Preprocessing typically involves the following steps:

1. **Data Cleaning:** This step involves handling missing or incorrect data by either imputing missing values or removing instances with missing values. Outliers can also be identified

and dealt with in this step.

2. **Data Transformation:** This step involves converting the data into a more suitable format for analysis. This may involve feature scaling, normalization, and transformation of categorical variables into numerical variables.

3. **Feature Selection:** This step involves selecting the most relevant features for the analysis and model building. This helps to reduce the dimensionality of the data and improve the efficiency of the models.

4. **Data Reduction:** This step involves reducing the size of the dataset by sampling or summarizing the data. This can help to reduce the computational complexity of the models.

1.4.3 Model Selection

Model Selection is the process of choosing an appropriate machine learning algorithm or architecture for a particular problem. The goal of Model Selection is to find a model that can generalize well to new, unseen data and minimize the risk of overfitting or underfitting.

There are various factors to consider when selecting a model, including the size and complexity of the dataset, the type of problem (e.g., classification or regression), and the available computing resources. Some common approaches to Model Selection include:

1. **Train-Test Split:** This involves splitting the data into a training set and a validation set and evaluating the performance of different models on the validation set. This approach is simple but may lead to overfitting if the validation set is too small.

2. **Cross-Validation:** This involves dividing the data into multiple folds and evaluating the performance of different models on each fold. This approach provides a more reliable estimate of the model's performance but can be computationally expensive.

3. **Grid Search:** This involves trying out different combinations of hyperparameters for a given model and selecting the best combination based on a performance metric.

Ultimately, the goal of Model Selection is to find a model that achieves high accuracy while avoiding overfitting or underfitting. This requires careful consideration of the problem and dataset, as well as an understanding of the strengths and weaknesses of different machine learning models.

1.4.4 Model Training

Model training is a critical process in Deep Learning, where a model is trained to recognize patterns and make predictions from input data by adjusting the model's weights and biases using a loss function and an optimization algorithm. The data is divided into training and validation sets, and the model is initialized with random weights and biases. The training data is fed into the model, and the output is compared to the actual output using the loss function. The weights and biases are updated to minimize the loss using an optimization algorithm such as stochastic gradient descent (SGD). The training process is repeated multiple times, with each iteration called an epoch, and the validation set is used to evaluate the model's performance after each epoch. The training process is stopped when the model's performance on the validation set no longer improves. Proper model training is crucial to a model's ability to generalize and make accurate predictions on new and unseen data.

1.4.5 Model Evaluation

Model evaluation is the process of assessing the performance of a machine learning model. It involves measuring the model's accuracy, precision, recall, and other metrics on a validation set that was not used during training. The purpose of model evaluation is to determine whether the model is generalizing well to new data and to identify any areas for improvement. Evaluation metrics can vary depending on the problem and the type of model used. Common evaluation techniques include cross-validation, hold-out validation, and A/B testing. Model evaluation is critical for ensuring that a machine learning model is robust and reliable in real-world scenarios.

1.4.6 Hyperparameter Tuning

Hyperparameter tuning is the process of selecting the optimal hyperparameters for a machine learning model. Hyperparameters are parameters that are set before training the model and cannot be learned during training. Examples of hyperparameters include learning rate, regularization strength, and number of hidden layers.

Hyperparameter tuning involves exploring different combinations of hyperparameters and evaluating the model's performance on a validation set. This can be done manually or using automated techniques such as grid search or random search. The goal of hyperparameter tuning is to find the combination of hyperparameters that results in the best performance on the validation set. Proper hyperparameter tuning can significantly improve the performance of a machine learning model.

1.4.7 Model Deployment

Model deployment refers to the process of deploying a trained machine learning model into a production environment where it can be used to make predictions on new data. This involves integrating the model with other software systems, setting up infrastructure to handle the model's computation and storage requirements, and ensuring that the model's performance meets the requirements of the production environment. Model deployment also involves ongoing monitoring and maintenance of the deployed model to ensure that it continues to perform well over time. Proper model deployment is essential for ensuring that the model is effective in real-world scenarios and can provide meaningful insights or predictions to end-users.

1.5 Technologies Used

1.5.1 Tensorflow

TensorFlow is an open-source machine learning framework developed by Google. It is widely used in the development of deep learning models, including neural networks and other machine learning algorithms. TensorFlow allows developers to define and train complex models using a high-level API in Python, making it easy to experiment with different models and architectures. TensorFlow is designed to be scalable and can be used on a variety of devices, including CPUs, GPUs, and specialized hardware like TPUs. TensorFlow has a large and active community of developers who contribute to its development and offer support to new users.

1.5.2 Keras

Keras is an open-source deep learning library written in Python. It is designed to provide a user-friendly interface for building and training deep neural networks. Keras allows for rapid prototyping and supports both convolutional and recurrent neural networks. Keras is built on top of other popular deep learning libraries such as TensorFlow, Theano, and CNTK, and it provides a high-level API that simplifies the process of building and training deep learning models. Keras has gained popularity due to its ease of use, flexibility, and excellent documentation. It is widely used in academia and industry for a variety of applications, including computer vision, natural language processing, and speech recognition.

1.5.3 OpenCV

OpenCV (Open Source Computer Vision) is an open-source computer vision and machine learning library that provides various tools and algorithms for image and video processing. It was initially developed by Intel and later supported by Willow Garage and Itseez. OpenCV supports various programming languages such as C++, Python, and Java, making it widely accessible. It includes functions for image processing, object detection, and tracking, facial recognition, and machine learning. OpenCV is widely used in research and industry, including applications in robotics, surveillance, medical imaging, and self-driving cars. Its rich functionality and broad community support make it a valuable tool for anyone working with computer vision and image processing.

1.5.4 MediaPipe

MediaPipe is an open-source cross-platform framework for building real-time machine learning applications. It provides a pipeline for processing media data such as video and audio streams and allows for the integration of machine learning models for tasks such as object detection, face detection, and hand tracking. MediaPipe includes pre-built components for common computer vision tasks and supports a variety of input sources such as webcams, microphones, and pre-recorded videos. The framework is built on top of TensorFlow and is designed to be modular and flexible, allowing developers to easily customize and extend its functionality. MediaPipe has been used in a variety of applications such as augmented reality, gesture recognition, and robotics.

1.5.5 gTTS

gTTS (Google Text-to-Speech) is a Python library that allows users to convert text to speech using Google's text-to-speech API. With gTTS, users can easily convert any text to a spoken audio file in a variety of languages and voices. The library is easy to use, and it supports a range of options such as language selection, voice selection, and speed control. gTTS is an open-source library and is widely used for generating speech for various applications, such as text-to-speech systems, chatbots, and audiobook production. It is a powerful tool for developers looking to add text-to-speech functionality to their applications with minimal effort.

1.6 Problem Statement

Translate the sign language used by speech and hearing-impaired people into English speech with the help of deep learning approaches.

CHAPTER - 2

LITERATURE SURVEY

CHAPTER - 2
LITERATURE SURVEY

2 Literature Survey

In this paper [1] the authors developed a trainable deep learning network for sign language recognition that can effectively capture spatiotemporal information with few frames of the signs. The hierarchical sign learning module comprises three networks: Dynamic Motion Network (DMN) learns the spatiotemporal information on the key frames of the sign gesture, Accumulative Motion Network (AMN) which accepts the AVM (bi-directionally fused) image as an input to learn the spatial information of this image, and Sign Recognition Network (SRN) which is basically a CNN takes the input as the concatenated features of DMN and AMN and consists the output as a SoftMax function. The proposed approach is evaluated on the KArSL and LSA64 datasets. KArSL is a multimodality ArSL dataset recorded using Kinect V2 at a rate of 30 frames per second which comprises 502 signs performed by three signers. LSA64 is an Argentinian sign language dataset that contains 3200 videos of 64 signs performed by ten signers.

This paper [2] introduces a new large-scale Word-Level American Sign Language (WLASL) video dataset, containing more than 2000 words performed by over 100 signers and compared different sign language recognition methods on the dataset including the holistic visual appearance approach with CNN and RNN, 2D human pose-based approach with GRU, and a novel pose-based temporal graph convolutional networks (Pose-TGCN) that model spatial and temporal dependencies in human pose trajectories simultaneously.

In this paper [3] the authors conducted a comprehensive review of automated sign language recognition based on machine/deep learning methods and techniques published between 2014 and 2021. The study indicates that input modalities bear great significance in this field; it appears that recognition based on a combination of data sources, including vision- based and sensor-based channels, is superior to a unimodal analysis. The study has addressed the complete pipeline including the data collection, feature extraction, feature selection and models used in sign language recognition.

In this paper [4] the authors proposed an approach that employed a 3D CNN and used transfer learning to beat the scarcity of a large labelled hand gesture dataset. The 3DCNN model learns region-based spatiotemporal features for hand gestures. The input of this model is a sequence of RGB frames captured by a basic camera. The model was evaluated on KSU-SSL, ArSL, and PURDUE RVL-SLLL datasets. The KSU-SSL contains 40 subjects recording the words and expressions in the Saudi sign language. ArSL contains 23 gestures performed by three participants. PURDUE RVL-SLLL consists of 43 classes of isolated hand gestures.

In this paper [5] the author optimizes the level of C3D architecture knowledge transfer between human activity recognition and hand gesture recognition. The authors present a hand gesture recognition system based on an optimized C3D architecture. The proposed system uses local and global configurations efficiently with more attention to the hand region. King Saud University Saudi Sign Language (KSU-SSL) dataset was used in the evaluation. The dataset contains isolated words and phrases from common expressions in the SSL dictionary. The dataset was recorded by 40 participants over five recording sessions without any restrictions.

This paper [6] reviewed the sign language research in the vision-based hand gesture recognition system from 2014 to 2020 with the objective to identify the progress and what needs more attention. The paper includes various techniques that are being used in Data Acquisition, Data Environments, Sign Languages Representations, and different feature extraction techniques.

In this paper [7] the authors proposed a CNN-based architecture for sign language alphabet recognition. The proposed finetuned CNN architecture contains multiple convolutional layers, max-pooling, dropout, and dense (fully connected). The MNIST dataset of the ASL in addition to the data augmentation is used as the input data.

In this paper [8] the authors in the first fold developed the SL recognition model using the Media Pipe library and the VGG-19 model. Furthermore, they incorporate the Bi-LSTM network for text generation. The incorporation of CNN and LSTM networks in such a hybrid way produces higher recognition accuracy and noticeable performance. The temporal details are analyzed sequentially to predict the translation text without any misclassification. The model is trained using the RWTH-PHOENIX-Weather 2014T dataset containing 40k videos for sentence-level German sign language and the ISL-CSLTR dataset which consists of 700 videos for 100 sentences each.

In this paper [9], the proposed method processed a video sequence of continuous words, by recognizing each word individually and achieving high accuracy. It proposes a method of backhand-view-based continuous-signed- letter recognition using a rewound video sequence with a previously signed letter. In this method, a hand shape of the previously signed letter and trajectories of finger joints moving from the previously signed letter to the current one is detected, features are then extracted, and finally, the features are classified for signed letter recognition. To evaluate the performance of the proposed method, experiments with 10 participants were performed 20 times each, and the results revealed 96.07improved significantly from the conventional methods using forehand and backhand.

In this paper [10], the B3D ResNet model is used to perform object location using a deep learning method, i.e., Faster R-CNN, to detect the hand and segment the hand position from the background. Then, the B3D ResNet model is used to extract the features of the inputted video sequence and analyze the feature sequence. By classifying the inputted video sequence, the hand gestures couldbe identified, and dynamic sign language recognition could be effectively achieved. SLR datasets are used and the B3D ResNet effectively recognized complex hand gestures through larger video sequence data and obtained high recognition accuracy for 500 vocabularies from Chinese hand sign language.

In this paper [11], the authors have presented a novel framework for continuous-SLR using a Leap motion sensor. A modified LSTM architecture and a fined tuned CNN architecture have also been proposed for the recognition of sign words and sentences. A dataset of 35 isolated sign words has been used while training the model. The evaluation of our approach has been performed on 942 signed sentences produced by six signers. Average accuracies of 72.3and isolated sign words, respectively.

In this paper [12], the authors developed a deep learning framework named SignBERT, integrating the bidirectional encoder representations from transformers (BERT) with the residual neural network (ResNet), to model the underlying sign languages and extract spatial features for CSLR. We further propose a multimodal version of SignBERT, which combines the input of hand images with an intelligent feature alignment, to minimize the distance between the probability distributions of the recognition results generated by the BERT model and the hand images. Experimental results indicate that when compared to the performance of alternative approaches for CSLR, our method has better accuracy with a significantly lower word error rate on three challenging continuous sign language datasets including the Hong Kong sign language dataset, RWTH-PHOENIX-Weather 2014 dataset and Chinese sign language dataset.

In this paper [13], the proposed system is based on the fast fisher vector (FFV) and bi-directional Long-Short Term memory (Bi-LSTM) method. Furthermore, comparison results demonstrate randomly selected ASL dictionaries and 10 pairs of similar ASL words, in leave-one subject-out cross-validation on the constructed dataset. A large database of dynamic sign word recognition algorithms called bidirectional long short-term memory-fast fisher vector (FFV-Bi-LSTM) is designed. The performance of our FFV-Bi-LSTM is further evaluated on the ASL data set, leap motion dynamic hand gestures data set (LMDHG), and Semaphoric hand gestures contained in the Shape Retrieval Contest (SHREC) dataset

In this paper [14] the proposed method of sign language recognition aims to detect the significant motions of the human body, especially the hands, analyze them and understand them. In their study, deep learning- based hand gesture recognition models are developed to accurately predict the emergency signs of Indian Sign Language. A total of three different models are used. Model I consist of 3D CNN, while Model II uses a combination of pre-trained CNN and LSTM. Model III, an object detection model, is based on the YOLO (You Only Look Once) v5 algorithm for detecting hand gestures. A video dataset based on words from Indian sign language. The dataset included eight emergency hand gestures representing ISL words.

In this paper [15] the authors proposed a novel spatiotemporal long- term memory (LTM) architecture that is motivated by neuro-biological evidence, such as hierarchical organization, fast learning, sparse connectivity, and error-tolerant retrieval. The ASL dataset which contains samples recorded by a high-quality hand position tracker from a native signer is used. The total number of signs is 95, each of which was recorded 27 times, and organized in nine different sessions. Each sample contains a 27-D temporal pattern of average length 57.

This paper [16], proposes an isolated sign language recognition model based on a model trained using Motion History Images (MHI) that are generated from RGB video frames. RGB-MHI images represent a Spatiotemporal summary of each sign video effectively in a single RGB image. The authors proposed two different approaches using this RGB-MHI model. In the first approach, we use the RGB-MHI model as a motion-based spatial attention module integrated into a 3D-CNN architecture. In the second approach, we use RGB-MHI model features directly with the features of a 3D-CNN model using a late fusion technique. The datasets used are the AUTSL contains 226 signs, and 36,302 video samples, and BosphorusSign22k another large-scale, isolated TSL dataset that contains 744 signs, and 22,542 video samples are used.

In this paper [17] the main objective is to identify a low-cost, affordable method that can facilitate hearing and speech-impaired people to communicate with the world in a more comfortable way where they can easily get what they need from society and can contribute to the well-being of the society. The proposed system starts by capturing the signs using the web camera which is interfaced with a laptop and processed without storing the recorded data. The image processing and segmentation are done and are stored for reference for real-time conversion of the processed images stored in the laptop for further reference.

In this paper [18] the study is designed in accordance with the PRISMA methodology to determine the technological advancements that are applied in sign language recognition, visualization, and synthesis. It identified image processing and deep learning are driving new applications and tools that improve the various performance metrics in these sign language-related tasks. Finally, the paper identified which techniques and devices contribute to such results and what are the common threads and gaps that would open new research directions in the field. It also provided the details of visualization such as the 3D model hand skeleton methods.

In this paper [19], the author tried to improve keyframe-centered clips (KCC) sampling to get a new kind of sampling method called optimized keyframe-centered clips (OptimKCC) sampling to select key actions from sign language videos. Besides, the design of a skeletal feature called Multi- Plane Vector Relation (MPVR) to describe the video samples is used. Finally, combined with the attention mechanism, the author also use Attention-Based networks to distribute weights to the temporal features and the spatial features extracted from skeletal data. The DEVISIGN sign language dataset includes 500 sign language words and uses Kinect-1.0 to capture RGB, depth, and skeletal data and the CSL dataset contains 200 sign language words, which are collected by Kinect-2.0 are used.

In this paper [20] the proposed system is based on a multilevel architecture that allows modeling and managing of the knowledge of the recognition process in a simple and robust way. The final abstraction level of this architecture introduces the semantic context and the analysis of the correctness of a sentence given in a sequence of recognized signs. A SOM neural network is used to classify the single sign and to provide a list of probable meanings. Experimentations are presented using a set of signs from the Italian sign language (LIS).

CHAPTER - 3
SYSTEM ANALYSIS

CHAPTER - 3
SYSTEM ANALYSIS

3 System Analysis

3.1 Existing System

The existing system uses a 3D convolutional neural network (3DCNN) for recognizing hand gestures in sign language. This approach uses a 3D convolutional neural network (CNN) to learn spatiotemporal features from a sequence of depth maps of hand gestures. The architecture of the existing system is depicted in Fig 3 with each module functions being described in below sections.

3.1.1 Image Cropping and Normalization

The existing system uses a sequence of RGB frames of different lengths as input videos, which are normalized using linear sampling to select only 16 frames from each video sequence. The sequence order is preserved during this normalization step to preserve highly discriminative features for gesture recognition.

Two cropping and normalization methods are then performed simultaneously on the selected frames. The first method locates the signer's face and estimates the gesture space to crop each frame based on the detected facial length and body parts ratios information. Each frame is resized to a fixed size of 112x112 pixels while preserving the aspect ratio. This method outputs a sequence of 16 frames, where each frame includes the entire gesture space.

The second method focuses more on the fingers' configuration by cropping and normalizing the hand region to reduce the effects of non-relevant features in each frame. This spatial normalization and cropping also help to avoid the effects of variations in the signer's height and distance from the camera.

Overall, the existing system architecture uses efficient hand gesture representation and spatial normalization and cropping techniques to improve the accuracy of sign language gesture recognition.

3.1.2 C3D

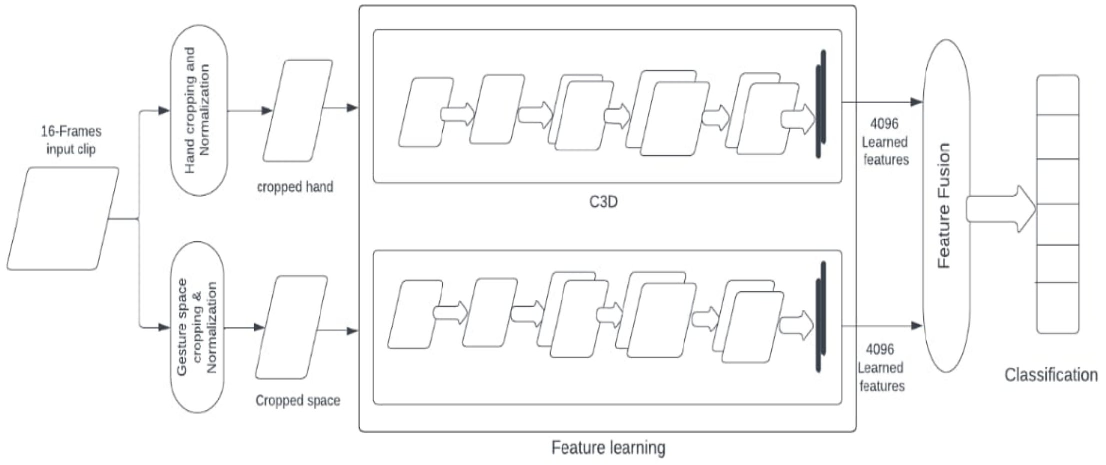


Fig 3: Architecture of the Existing System

The C3D architecture is a 3D convolutional neural network that extends the concept of 2D convolutional neural networks to video data. It consists of 8 convolutional layers, 5 pooling layers, and 2 fully connected layers. The network takes a 3D input volume, where the third dimension corresponds to time.

The convolutional layers use 3D kernels to extract spatiotemporal features from the video data. The pooling layers perform 3D max-pooling to reduce the dimensionality of the feature maps. The fully connected layers are used for classification and produce the output probabilities for each class.

C3D was initially designed for action recognition in videos and was trained on the Sport-1M dataset, which contains over a million videos of sports-related activities. However, it has since been used in various applications, including hand gesture recognition, where it has shown promising results.

3.1.3 Feature Learning and Classification

The existing system proposes a deep learning-based approach for sign language gesture recognition with an efficient hand gesture representation. The existing approach starts with a pre-trained C3D architecture that has eight convolutional layers, five pooling layers, and two fully connected (FC) layers. They replace the last block of the model, which has two FC layers, with a new FC layer of 4096 neurons to reduce the training cost. The authors optimize the level of knowledge transfer from the source domain to the target domain to represent spatiotemporal features at different levels of the frame sequence.

Two instances of the optimized C3D architecture are used to represent the spatiotemporal features in different levels of the frame sequence, i.e., the hand region and the entire gesture space region. The first C3D instance learns the fine spatiotemporal features of the hand configuration, where the hand is dominant in each input frame. The second C3D instance learns the coarse spatiotemporal features of the whole-body configuration. This phase produces two feature vectors, each having a dimension of 4096.

The extracted 4096 features from both the hand and gesture represented frames are combined using the MLP and autoencoder feature fusion techniques. The output of the feature fusion techniques is given to the SoftMax layer which outputs the probabilities corresponding to each of the classes.

3.1.4 Drawbacks

1. The system requires a large amount of training data, which can be difficult to collect for some sign languages.
2. The hand-cropping and normalization process may not work well for signers with different body proportions or clothing.
3. The C3D network used in the architecture is quite complex and may require significant computational resources for training and inference.
4. It only considers a single hand in the image, ignoring the possibility of sign language gestures that involve both hands.

3.2 Proposed System

3.2.1 Description

The proposed system aims to bridge the communication gap between speech and hearing-impaired individuals and the rest of society by using deep learning approaches. The system leverages spatiotemporal features of sign language gestures to identify appropriate English speech for each gesture. To accomplish this, sign language gesture videos from the WLASL dataset are preprocessed into a set of 30 frames with relative timestamps between any two frames being constant. MediaPipe's pose detection technology is then utilized to identify the hand, pose, and facial landmarks for each frame in the set.

The set of extracted landmarks is then used to train the GRU model, which is a variant of RNN. The training step incorporates the Keras sequence class, which makes the training of large datasets memory efficient. The GRU model outputs probabilities that the video corresponds to each of the English words. Finally, the Google gTTS library is used to produce the speech output of the given sign language gesture.

This system not only relies on deep learning approaches but also incorporates advanced technologies such as pose detection to enhance the accuracy of the model. By leveraging spatiotemporal features of sign language, the system can provide a more intuitive and natural way of communication for speech and hearing-impaired individuals. The use of the GRU model ensures that the system can handle long-term dependencies that are common in sign language gestures, and the Keras sequence class ensures that the system can handle large datasets without running into memory issues.

3.2.2 Advantages

The advantages of the proposed system are:

1. **Accurate translation:** By using deep learning approaches to extract spatiotemporal features of sign language, the proposed system may be able to more accurately identify the appropriate English speech for each gesture.

2. **Efficient training:** The use of Keras sequence class in the training step of the GRU model

can help make the training of large datasets more memory efficient.

3. **Scalability:** The use of MediaPipe's pose detection allows the system to identify the hand, pose, and facial landmarks in each frame of the sign language videos, which could make it easier to scale the system up to handle larger datasets or more complex sign language gestures.

4. **Speech output:** By using Google's gTTS library to produce the speech output of the sign language gestures, the proposed system could provide a convenient way for speech and hearing-impaired individuals to communicate with others who do not know sign language.

5. **Language independence:** The system can be trained on any sign language dataset and can be used to translate sign language gestures into speech in any language. This means that the system can be adapted to different cultures and regions, making it a more versatile solution for speech translation.

3.3 Hardware Requirements

1. **RAM:** 8 GB
2. **Hard Disk:** 100 GB
3. **Processor:** Intel Core i7 or i9 processor or an AMD Ryzen 7 or 9 processor
4. **Speed:** 2 GHz or higher
5. **Audio Player:** Any MP3 Player

3.4 Software Requirements

3.4.1 Python

Python is a high-level programming language widely used in deep learning due to its simplicity, readability, and availability of numerous libraries and frameworks. Python provides a wide range of libraries for deep learning, including TensorFlow, Keras, PyTorch, and MXNet. These libraries offer a comprehensive set of APIs for building deep learning models with ease, reducing the need for low-level programming. Python's syntax allows developers to write concise and expressive code that can be easily maintained and extended. Additionally, Python has a large and active community that constantly contributes to the development of new libraries and tools, making it easier to work with deep learning. With its ease of use and versatility, Python has become the language of choice for many deep learning applications.

3.4.2 OpenCV

OpenCV (Open Source Computer Vision) is a computer vision and machine learning library that is open source and offers several algorithms and tools for video and image processing. It was initially developed by Intel and later supported by Willow Garage and Itseez. OpenCV is widely used in industry and research for applications such as machine learning, facial recognition, object detection and tracking, and medical imaging. It is also versatile, being compatible with C++, Python, and Java. OpenCV's broad community support and rich functionality make it a valuable tool for anyone working with image processing and computer vision.

3.4.3 MediaPipe

MediaPipe is an open-source framework developed by Google that provides various tools for building multimedia pipelines. It enables developers to create complex pipelines for processing various forms of media, such as video and audio, with support for real-time processing. MediaPipe offers a wide range of pre-built components for various tasks, such as object detection, face detection, hand tracking, pose estimation, and more. Additionally, MediaPipe allows users to create custom pipelines using pre-built modules, which can be combined to create more complex pipelines. MediaPipe has gained popularity in various fields, including robotics, augmented reality, virtual reality, and self-driving cars, due to its powerful functionality and ease of use.

3.4.4 Tensorflow

Google developed TensorFlow, an open-source machine learning framework. It is a popular choice for deep learning models, including neural networks and machine learning algorithms.

TensorFlow offers a high-level Python API, making it easier for developers to define and train complex models. This flexibility allows developers to experiment with different models and architectures. TensorFlow is scalable and can work on a range of devices, such as CPUs, GPUs, and TPUs. Additionally, TensorFlow has a large and active community of developers who contribute to its development and provide support to new users.

3.4.5 Keras

Keras is a high-level neural network API written in Python. It is designed to enable fast experimentation with deep neural networks and focuses on being user-friendly, modular, and extensible. Keras supports multiple backends, including TensorFlow, CNTK, and Theano, allowing users to choose the best backend for their specific hardware configuration. It provides a simple and intuitive interface for building and training various types of deep learning models, including convolutional neural networks (CNNs), recurrent neural networks (RNNs), and others. Keras also offers various pre-trained models, making it easy for users to leverage pre-trained models for various tasks such as image classification, object detection, and natural language processing.

3.4.6 gTTS

gTTS (Google Text-to-Speech) is a Python library and a command-line utility that uses Google's Text-to-Speech API to convert text into a natural-sounding voice in various languages. It supports multiple languages and allows users to choose from a variety of available voices with different characteristics. gTTS is easy to use and can be integrated into various Python applications, including text-to-speech systems, voice assistants, and automated narration. It can also generate audio files in different formats, including MP3, WAV, and Ogg Vorbis. gTTS is a powerful tool for developers who want to add voice capabilities to their applications without the need for pre-recorded audio files or expensive text-to-speech software.

3.4.7 Matplotlib and Seaborn

Matplotlib and Seaborn are popular data visualization libraries in Python used for creating high-quality graphs, plots, and charts. Matplotlib is a versatile library that offers a wide range of visualization options and is commonly used for creating basic plots such as line graphs, scatter plots, histograms, and bar charts. Seaborn is built on top of Matplotlib and provides a higher-level interface for creating more complex and aesthetically pleasing visualizations. It offers additional features such as heat maps, cluster maps, and violin plots, and allows for easy customization of colors and styles. Both Matplotlib and Seaborn are widely used in the field of data science for exploratory data analysis and presentation of results.

CHAPTER - 4

ARCHITECTURE OF THE PROPOSED SYSTEM

CHAPTER - 4

ARCHITECTURE OF THE PROPOSED SYSTEM

4 Architecture of the Proposed System

The architecture of the proposed system involving the process of generating speech corresponding to the gesture video is represented in Fig 4.

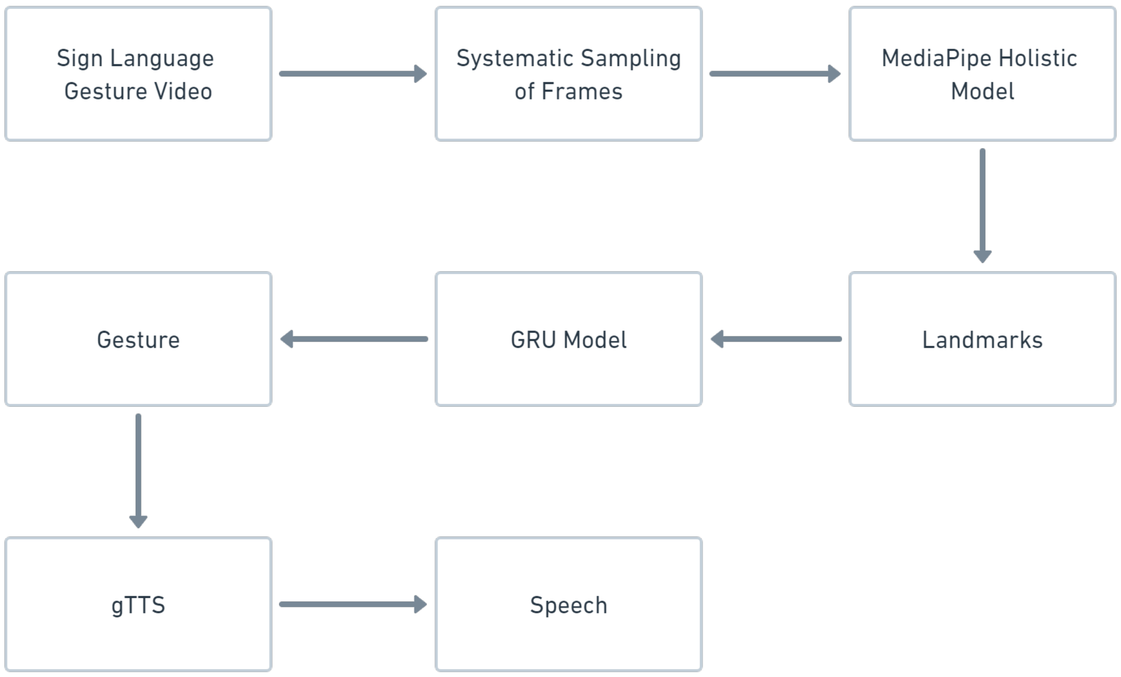


Fig 4: Architecture of the Proposed System

4.1 Extracting Landmarks

Using the systematic sampling technique, a set of 30 frames is extracted from each video. The frames are carefully selected to ensure that the interval between any two adjacent frames is constant.

The MediaPipe Holistic model is an AI-based computer vision model that provides real-time detection and tracking of facial features, hand landmarks, and full-body posture. It uses machine learning and deep neural networks to accurately identify and track key points in the human body. This MediaPipe Holistic model is used to extract the face, hand and pose landmarks from each of the sampled video frames. As the final output the MediaPipe Holistic model outputs 33 pose landmarks, 468 face landmarks, and 21 hand landmarks per hand. These landmarks for the set of 30 frames are given as the input while training the GRU model.

4.2 GRU Model

4.2.1 GRU

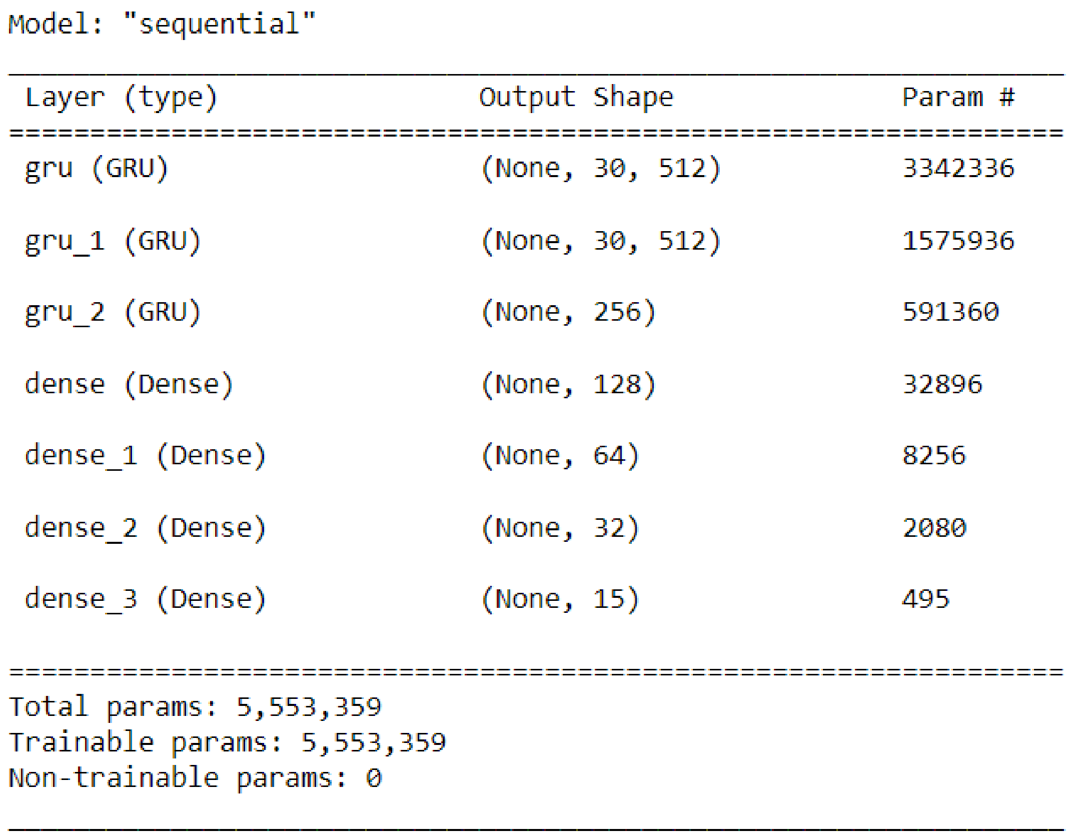
Gated Recurrent Unit (GRU) is a type of Recurrent Neural Network (RNN) that is designed to address the vanishing gradient problem in traditional RNNs. GRU is a variant of the Long Short-Term Memory (LSTM) architecture, which is another type of RNN that has been successful in modeling sequential data.

GRUs have fewer parameters than LSTMs, making them faster to train and less prone to overfitting. The main innovation in the GRU architecture is the introduction of two new gates, the reset gate and the update gate, which control the flow of information within the network. The reset gate determines which part of the past information should be forgotten, while the update gate determines which part of the current input should be added to the memory.

One of the main advantages of using GRUs is their ability to capture long-term dependencies in sequential data, which makes them particularly suitable for modeling time-series data. Moreover, GRUs have shown promising results in applications where the amount of training data is limited or where real-time processing is required.

4.2.2 Model Architecture

The model, which is shown in Fig 5, comprises three GRU layers with 512, 512 and 256 units in the respective layers. Both the recurrent dropout and dropout between the layers of the model are applied during the training to avoid the risk of overfitting. The GRU layers are followed by three Dense layers with 128, 64 and 32 neurons respectively. All the layers have the activation as ReLU. The output layer is a SoftMax layer with 15 neurons that output the probabilities of



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 gru (GRU)                   (None, 30, 512)           3342336

 gru_1 (GRU)                 (None, 30, 512)           1575936

 gru_2 (GRU)                 (None, 256)               591360

 dense (Dense)               (None, 128)               32896

 dense_1 (Dense)             (None, 64)                8256

 dense_2 (Dense)             (None, 32)                2080

 dense_3 (Dense)             (None, 15)                495

=================================================================
Total params: 5,553,359
Trainable params: 5,553,359
Non-trainable params: 0
_________________________________________________________________
```

Fig 5: Layer-by-Layer Overview of the Trained Deep Learning Model

each gesture corresponding to one of the 15 words.

GRUs have fewer parameters than LSTMs, making them faster to train and less prone to overfitting. The main innovation in the GRU architecture is the introduction of two new gates, the reset gate and the update gate, which control the flow of information within the network. The reset gate determines which part of the past information should be forgotten, while the update gate determines which part of the current input should be added to the memory.

One of the main advantages of using GRUs is their ability to capture long-term dependencies in sequential data, which makes them particularly suitable for modeling time-series data. Moreover, GRUs have shown promising results in applications where the amount of training data is limited or where real-time processing is required.

4.3 Speech Generation

The words are selected for speech corresponding to each video based on the probabilities given by the GRU model. gTTS (Google Text-to-Speech) library is used for generation of speech. gTTS is a Python library that can convert text to speech using Google's Text-to-Speech API. To generate speech using gTTS, first the text is prepared by concatenating the words predicted by model. Then, the gTTS library is used to generate an audio file (.mp3 or .wav) from the prepared text. The generated audio file can be saved or played back directly in the Python program. This process enables the model to output speech as a final result, allowing it to be used for tasks such as automatic transcription of sign language videos or speech generation.

CHAPTER - 5
SYSTEM IMPLEMENTATION

CHAPTER - 5

SYSTEM IMPLEMENTATION

5 System Implementation

5.1 Data Preprocessing

5.1.1 Data

The WLASL (Word-Level American Sign Language) is used for training the model. WLASL dataset is a large-scale sign language recognition dataset that contains videos of people performing over 2000 sign language words. The dataset was created to address the lack of available sign language recognition datasets, particularly those at the word level. The videos were collected from a diverse range of signers and environments to ensure the dataset's broad applicability. In addition to the videos, the dataset includes annotations for the sign language words, as well as information about the signer's gender and handedness. The WLASL dataset has been used to train and evaluate various sign language recognition models, making significant contributions to the field of computer vision and machine learning.

Some of the important columns in the dataset include:

1. **Gloss:** This column contains the text label for the sign language gesture being performed in the video.
2. **Video ID:** This column contains a unique identifier for each video sample in the dataset.
3. **Signer:** This column identifies the person performing the sign language gesture in the video.
4. **Signer ID:** This column contains a unique identifier for each signer in the dataset.
5. **Annotation:** This column contains additional information about the video sample, such as whether the gesture is one-handed or two-handed, or whether the signer is performing the gesture in a standing or sitting position.

6. **Frames:** This column contains the total number of frames in the video sample.

5.1.2 Preparing the Videos for Model Training

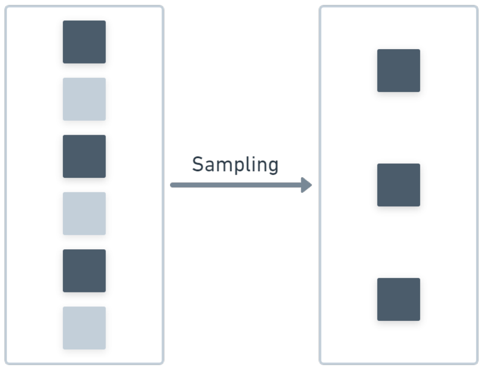


Fig 6: Sampling of the Video Frames

The process of systematic sampling is utilized to extract a group of 30 frames from every video, ensuring that the frames are chosen in a manner that maintains a uniform time interval between each successive selected frame. This process is illustrated in the Fig 6.

The MediaPipe Holistic model is an advanced computer vision model that uses machine learning and deep neural networks to detect and track various facial features, hand landmarks, and body posture in real-time. It is used to extract the pose, hand, and face landmarks from the sampled video frames. The model generates 33 pose landmarks, 468 face landmarks, and 21 hand landmarks per hand, which are used as inputs while training the GRU model.

5.1.3 One-Hot Encoding

One hot encoding is a technique used to represent categorical data as numerical data that can be used in machine learning models. In one hot encoding, each category is represented by a binary vector of 0s and 1s, where the length of the vector is equal to the total number of categories. In this vector, only one element corresponding to the category is 1, and all other elements are 0.

For example, if we have three categories A, B, and C, we would represent them as [1,0,0], [0,1,0], and [0,0,1] respectively. This allows machine learning models to use categorical data as input since the binary vector can be treated as numerical data.

Labels are one-hot encoded to convert them into a format suitable for training the GRU model. By encoding the labels, the model can better understand and differentiate between different categories, leading to more accurate predictions.

5.2 Model

5.2.1 LSTM vs GRU

Long Short-Term Memory (LSTM) is a type of recurrent neural network (RNN) architecture that is designed to handle the vanishing gradient problem and can effectively capture long-term dependencies in sequential data. It has a memory cell that allows information to be stored for long periods of time and selectively erased or updated based on the input, and also has gates that regulate the flow of information.

GRU, or Gated Recurrent Unit, is a type of Recurrent Neural Network (RNN) that uses gating mechanisms to selectively update and reset information in the hidden state. It is designed to address the issue of vanishing gradients in long sequences, making it useful for processing sequential data such as natural language and speech. GRU is faster and more efficient than other RNN variants but its performance depends on the specific task and dataset.

GRU is a simpler and more efficient version of LSTM as it combines the forget and input gates into a single update gate, and the output gate is removed. This results in fewer parameters to train and thus faster convergence, while still maintaining comparable performance to LSTM on many tasks.

5.2.2 Gating Mechanism of GRU

GRU (Gated Recurrent Unit) is a type of recurrent neural network that can remember and forget information over long sequences. It has a gating mechanism that controls the flow of information through the network, allowing it to selectively remember or forget information as represented in Fig 7.

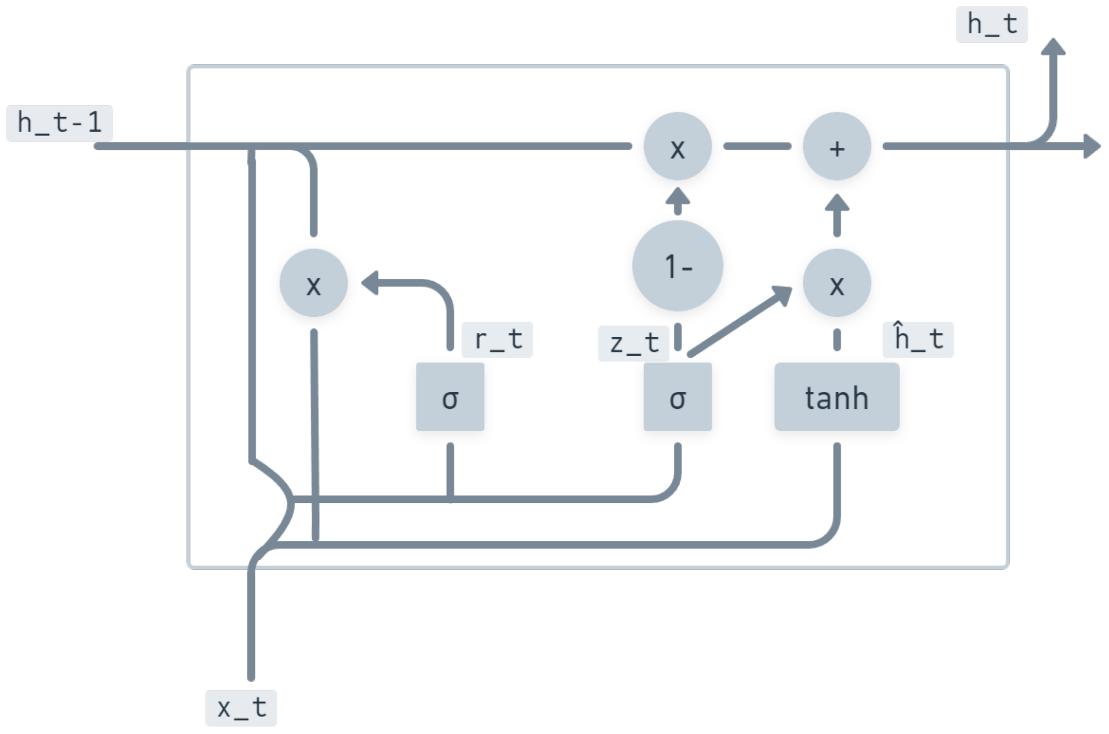


Fig 7: The Anatomy of the GRU Gating Mechanism

The equations for the update and reset gates in a GRU are:

$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r) \quad \ldots \quad (1)$$

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z) \quad \ldots \quad (2)$$

Here, x_t is the input at time step t, h_{t-1} is the hidden state from the previous time step, W_z, U_z, W_r, U_r are weight matrices, b_z, b_r are bias vectors, and σ is the sigmoid function.

The update gate z_t in Equation 2 controls how much of the previous hidden state h_{t-1} should be passed to the current state, while the reset gate r_t in Equation 1 controls how much of the previous hidden state h_{t-1} should be ignored.

The candidate hidden state $\tilde{h}_t$ in Equation 3 is computed using the input and the reset gate:

$$\tilde{h}_t = \tanh(W x_t + r_t \odot U h_{t-1} + b) \quad \ldots \quad (3)$$

Here, $\odot$ denotes element-wise multiplication, and tanh is the hyperbolic tangent function.

Finally, the new hidden state h_t Equation 4 is computed as a weighted combination of the previous hidden state and the candidate hidden state:

$$h_t = (1 - z_t) \odot \tilde{h_t} + z_t \odot h_{t-1} \quad (4)$$

The gate values and candidate hidden state are computed based on the current input and previous hidden state, allowing the network to selectively update or retain information over time.

5.2.3 Keras Sequence

Keras Sequence is a utility provided by the Keras deep learning framework that allows developers to easily and efficiently work with large datasets during model training. It is a Python generator that enables loading of large datasets in smaller, more manageable batches while training the model. Keras Sequence provides an efficient way of loading the data in the background while the model is being trained. The major features of Keras Sequence include

1. **Efficient data loading:** It loads the data efficiently in small batches while training the deep learning models.
2. **Supports parallelism:** Keras Sequence supports parallelism by allowing the loading of data on multiple CPU cores, thereby reducing the training time.
3. **Supports data augmentation:** It allows for data augmentation techniques such as random cropping, flipping, and rotating to be applied to the training data.
4. **Compatible with Keras models:** Keras Sequence is compatible with all Keras models, allowing seamless integration into any Keras-based deep learning workflow.
5. **Flexibility in input data types:** Keras Sequence provides flexibility in input data types, including images, text, and time-series data, among others.
6. **Easy to customize:** Keras Sequence is easy to customize and can be tailored to specific use cases and data formats.

5.2.4 ReLU

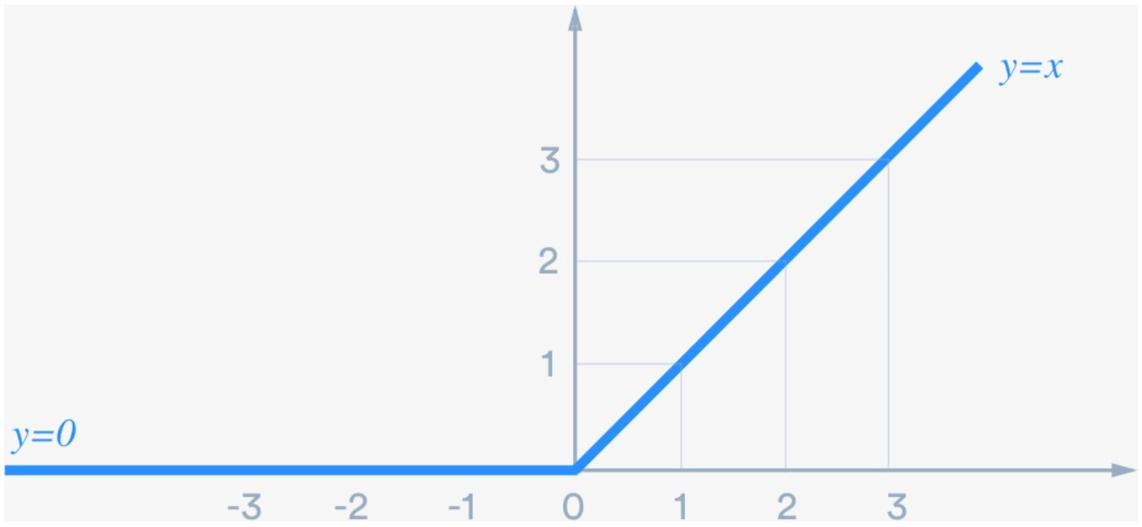


Fig 8: ReLU Activation Function

ReLU, short for Rectified Linear Unit, is an activation function commonly used in deep learning neural networks. It is a simple and computationally efficient function defined as: $f(x) = \max(0,x)$. The graph of the ReLU as depicted in Fig 8.

The function takes an input value x and returns either 0 if x is negative or x if x is non-negative. This means that ReLU is a non-linear function that introduces non-linearity into the neural network, which is essential for learning complex relationships between input and output. The key advantage of ReLU over other activation functions like sigmoid and tanh is that it is less susceptible to the vanishing gradient problem that can occur during backpropagation in deep neural networks. This is because ReLU has a simple derivative that is either 0 or 1, making it easier to compute and less likely to result in a vanishing gradient. ReLU has become the most popular activation function in deep learning and is used in many state-of-the-art models.

The main features ReLU include:

1. **Computational efficiency:** ReLU is computationally efficient to compute as it only involves simple thresholding operations. This makes it faster than other activation functions such as sigmoid and tanh.

2. **Non-saturating:** Unlike sigmoid and tanh, ReLU does not saturate in the positive region,

which means that it can learn faster and better in deep neural networks.

3. **Sparse activation:** ReLU produces sparse activations, which means that it can result in more efficient and effective feature representations.

4. **Robustness to vanishing gradients:** ReLU is less susceptible to the vanishing gradient problem, which can occur during training of deep neural networks.

5. **Interpretability:** ReLU produces interpretable outputs as it simply thresholds the input, making it easier to understand and interpret the behavior of the network.

5.2.5 Dropout

Dropout is a regularization technique used to prevent overfitting in neural networks. It works by randomly dropping out (setting to zero) a certain percentage of neurons during training, which forces the network to learn more robust and generalizable features. This helps prevent the network from relying too heavily on specific features of the training data that may not generalize well to new data. Dropout can be applied to any layer in the network, including input, hidden, and output layers. The dropout rate is a hyperparameter that controls the percentage of neurons to drop out, typically ranging from 0.1 to 0.5. Dropout is a widely used technique in deep learning and has been shown to improve performance on a variety of tasks.

Recurrent dropout is a technique used to prevent overfitting in recurrent neural networks (RNNs) such as GRUs and LSTMs. It involves randomly dropping out some of the neurons or hidden units in the network during training. This encourages the network to learn more robust and generalizable representations, rather than memorizing specific examples from the training data. Recurrent dropout differs from regular dropout in that it also drops out connections between the recurrent hidden states. This helps prevent overfitting in the temporal dimension, as it prevents the network from relying too heavily on previous time steps for predictions. Recurrent dropout can be applied in various ways, such as applying the same dropout mask at each time step, or using a different mask for each time step.

5.2.6 GRU Training

The model includes three GRU layers, each with different units: 512, 512, and 256. During training, the model uses both recurrent and dropout to avoid overfitting. Three Dense layers with 128, 64, and 32 neurons follow the GRU layers, with ReLU as the activation function. The final layer is a SoftMax layer with 15 neurons, which outputs the probabilities of each gesture corresponding to the 15 words.

5.3 Speech Generation

In the first step, the model computes the probabilities of each of the 15 words for every video in the given folder. The word with the highest probability is assigned as the label for that video. In the second step, the assigned word labels are used to form a sentence based on the order of the videos in the folder. Finally, the sentence is converted to speech using the gTTS (Google Text-to-Speech) library, which generates an audio file from the given text. This process allows for the automatic generation of spoken sentences from a sequence of videos, which could be useful in applications such as automated video captioning or speech-enabled video search.

CHAPTER - 6

WORKFLOW OF THE SYSTEM

CHAPTER - 6

WORKFLOW OF THE SYSTEM

6 Workflow of the System

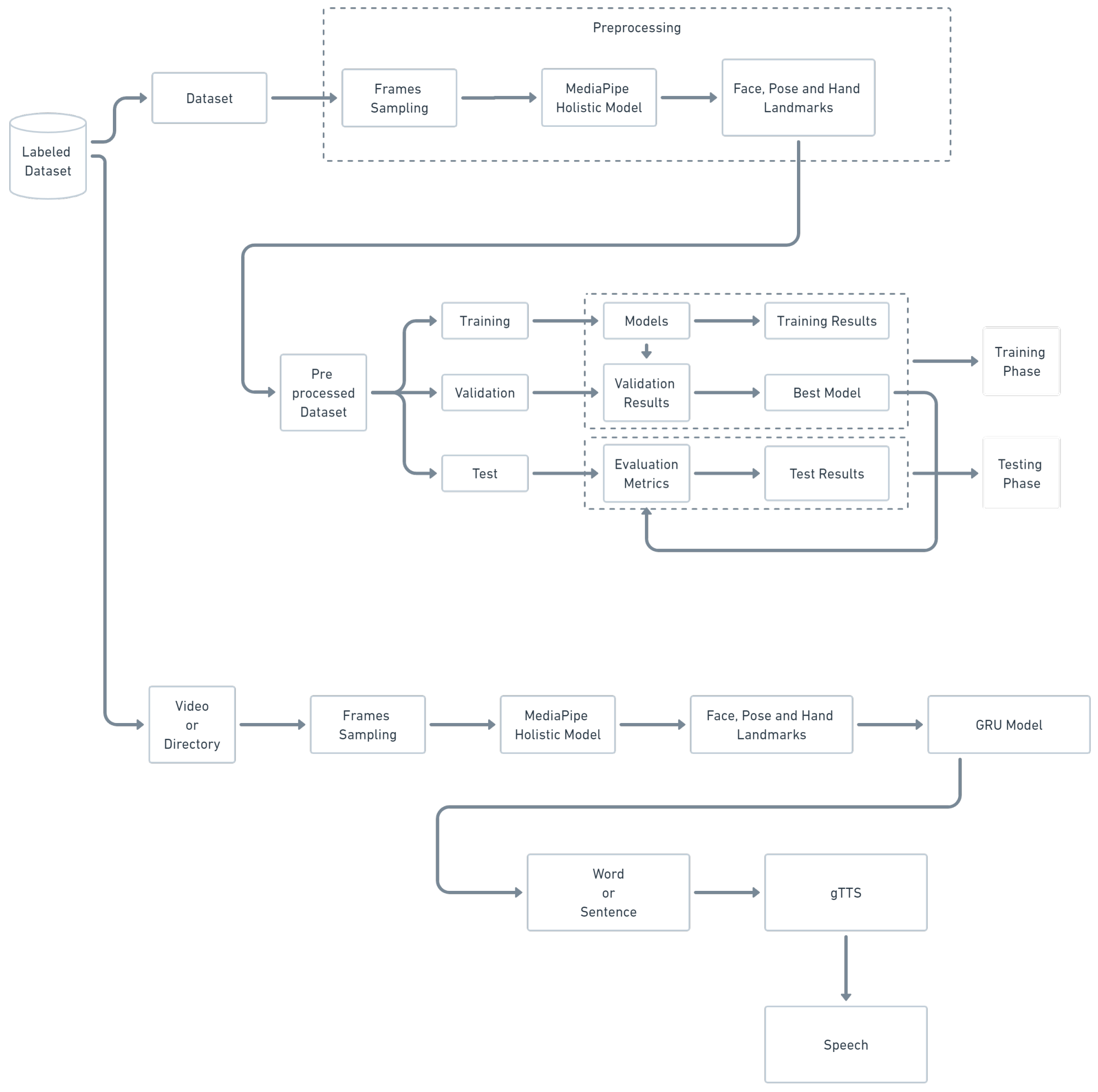


Fig 9: Workflow of the Proposed System

The workflow involves the following steps as delineated in Fig 9:

1. **DataSet of Videos:** The first step in this workflow is to obtain a dataset of videos containing various hand gestures. This dataset should be diverse and representative of the gestures that the model is expected to recognize.

2. **Systematic Sampling of 30 frames from the video:** Since videos contain a large amount of data, it is not feasible to process every frame of the video. Therefore, a systematic sampling technique is used to extract a set of 30 frames from each video. These frames are carefully selected to ensure that the interval between any two adjacent frames is constant.

3. **Retrieve pose, hand, face landmarks from the MediaPipe Holistic Model:** The MediaPipe Holistic model is an AI-based computer vision model that provides real-time detection and tracking of facial features, hand landmarks, and full-body posture. This model is used to extract the face, hand, and pose landmarks from each of the sampled video frames. The output of the model includes 33 pose landmarks, 468 face landmarks, and 21 hand landmarks per hand.

4. **Split the dataset into Train, Val, Test:** Once the landmarks are extracted from the videos, the dataset is split into three parts: training, validation, and testing. The training set is used to train the model, the validation set is used to tune hyperparameters and avoid overfitting, and the test set is used to evaluate the final performance of the model.

5. **Use the Train to train the GRU model Use the validation set to see whether the model is overfittng:** The GRU (Gated Recurrent Unit) model is a type of Recurrent Neural Network (RNN) that is used for processing sequential data. The training data is used to train the GRU model to recognize the hand gestures. During the training process, both the recurrent dropout and dropout between the layers of the model are applied to avoid the risk of overfitting. The model is trained for around 550 epochs, and the validation set is used to monitor the performance of the model and ensure that it is not overfitting.

6. **Analyze the performance of the model on the test set:** Finally, the performance of the model is evaluated on the test set. This is done by measuring the accuracy of the model in recognizing the hand gestures in the test videos. If the model performs well on the test set, it can be considered to be a successful gesture recognition system. If not, the model may

need to be retrained or the dataset may need to be modified to improve the performance.

6.1 Data

The WLASL dataset is a large-scale sign language recognition dataset consisting of videos of people performing over 2000 sign language words, with annotations for the sign language words, as well as information about the signer's gender and handedness. The dataset has been used to train and evaluate various sign language recognition models, contributing significantly to the field of computer vision and machine learning. Important columns in the dataset include Gloss, Video ID, Signer, Signer ID, Annotation, and Frames.

Among various Sign Language Gesture datasets the WLASL dataset is considered better than other datasets for Sign Language Recognition (SLR) due to several reasons. Firstly, it is a large-scale dataset with over 2000 sign language words, making it one of the largest datasets available for SLR. Secondly, the dataset is annotated at the word level, which is crucial for building models that can accurately recognize individual words. Thirdly, the dataset includes videos of signers from diverse backgrounds, genders, and ages, and it is captured in various environments, making it more representative of real-world scenarios. Finally, the dataset also includes information about the signer's handedness, which is useful for developing models that can recognize signs performed by both left-handed and right-handed signers. Overall, the WLASL dataset has contributed significantly to the field of SLR by providing a comprehensive and diverse dataset that can be used to train and evaluate a variety of sign language recognition models.

6.2 Data Preprocessing

The first step in the deep learning workflow is to gather and prepare the dataset. In this case, the dataset consists of videos of American Sign Language (ASL) gestures from the WLASL (Word-Level American Sign Language) dataset. The dataset contains over 2,000 sign language words and was created to address the lack of available sign language recognition datasets, particularly those at the word level. The videos were collected from a diverse range of signers and environments to ensure the dataset's broad applicability.

The second step is to systematically sample 30 frames from each video. This is done to reduce the amount of data while still capturing enough information for training the model. Systematic sampling ensures that the frames are evenly spaced out throughout the video, so important information is not missed.

The third step is to retrieve pose, hand, and face landmarks from the MediaPipe Holistic Model. This model is used to extract key points from the video frames, which represent the signer's body, hands, and face. These landmarks are important features that the GRU model will use to recognize the ASL gestures.

The final step is to split the dataset into train, validation, and test sets. The train set is used to train the GRU model, the validation set is used to monitor the model's performance and prevent overfitting, and the test set is used to evaluate the model's accuracy on unseen data. The dataset is split in a stratified manner, meaning that the proportion of signers and gestures is consistent across the train, validation, and test sets. This ensures that the model is trained on a representative sample of the data and can generalize to new signers and gestures.

6.3 Building the Model

During the training process, the GRU model is trained on the training set and validated on the validation set. The validation dataset is used to identify any potential overfitting patterns while training the model. The training is performed for around 550 epochs. To prevent overfitting, both recurrent dropout and dropout are used during the training process. Finally, the performance of the model is evaluated on the test set to assess the accuracy and effectiveness of the model in recognizing sign language gestures.

In the proposed GRU model architecture, there are three GRU layers added to the model. The first two layers have 512 units each with 'relu' activation and return sequences. The input shape for the model is set as 30 time steps and 1662 features.

The second GRU layer has recurrent dropout and dropout set to 0.4 to avoid overfitting during the training process. The third GRU layer has 256 units and returns only the last sequence. This layer also has recurrent dropout and dropout set to 0.6 to avoid overfitting.

After the three GRU layers, there are three dense layers added to the model, with 128, 64 and 32 neurons respectively, and all with the 'relu' activation function. The final output layer of the model is a dense layer with 15 neurons, each corresponding to one of the 15 gesture classes in the dataset, and using the ’softmax’ activation function to output the class probabilities.

6.4 Testing the Model

After training the GRU model using the training and validation sets, the performance of the model is evaluated on the test set. The purpose of testing is to assess the generalization ability of the model, i.e., how well it performs on unseen data.

To test the model, the test set is passed through the trained model, and the predicted outputs are compared with the ground truth labels. The performance metrics such as accuracy, precision, recall, and F1-score are calculated to evaluate the model’s performance.

The accuracy is calculated as the ratio of the correctly predicted samples to the total number of samples in the test set. Other metrics such as precision, recall, and F1-score provide more detailed information about the model’s performance. Precision measures the proportion of true positives among the predicted positives, while recall measures the proportion of true positives among the actual positives. The F1-score is the harmonic mean of precision and recall and provides a balance between the two.

In addition to the performance metrics, the confusion matrix is also used to evaluate the model’s performance. The confusion matrix is a table that summarizes the number of true positives, true negatives, false positives, and false negatives for each class in the test set. It provides valuable insights into the model’s performance, such as which classes are difficult to predict and which classes are often confused with each other.

Overall, this helps to ensure that the model can perform well on unseen data and is not overfitting to the training data. It also provides insights into the model’s strengths and weaknesses and helps to identify areas for improvement.

6.5 Using the Model

The following are the sequence of steps involved in using the model

Step 1: Video

The first step in the workflow is to have a video input containing the sign language gesture to be translated into speech. The video can be captured from any source, including cameras or pre-recorded videos.

Step 2: Systematic Sampling of 30 frames from the video

The second step involves systematic sampling of 30 frames from the video. This step ensures that an adequate number of frames are considered for the translation process. The number 30 is chosen as it is sufficient for capturing the necessary information in most cases.

Step 3: Retrieve pose, hand, face landmarks from the MediaPipe Holistic Model

The third step involves retrieving pose, hand, and face landmarks from the MediaPipe Holistic Model. This model is used to extract relevant features from the frames, which can then be used for sign language recognition. The model is based on a deep neural network that performs holistic detection of multiple modalities (pose, face, and hand) simultaneously.

Step 4: Use the trained GRU model to predict the corresponding word for the given Sign language Gesture

The fourth step involves using the trained GRU (Gated Recurrent Unit) model to predict the corresponding word for the given sign language gesture. The model has been trained on a large dataset of sign language gestures, which allows it to accurately recognize and translate new gestures. The input to the model is the extracted features from step 3.

Step 5: Use the gTTS to generate Speech

The final step in the workflow involves using gTTS (Google Text-to-Speech) to generate speech from the predicted word. gTTS is a Python library that converts text to speech using Google's Text-to-Speech API. Once the predicted word is obtained from step 4, it can be passed to gTTS to generate speech in the desired language and accent.

CHAPTER - 7
SOURCE CODE

CHAPTER - 7

SOURCE CODE

7 Source Code

7.1 Importing the Necessary Libraries

```
import tensorflow as tf
from tensorflow import keras
import json
import numpy as np
import cv2
import mediapipe as mp
import time
import os
import math
import matplotlib.pyplot as plt
import pickle
from tensorflow.keras.utils import to_categorical
from gtts import gTTS
from fractions import Fraction
```

7.2 Encoding Dataset Information from JSON to Python Dictionary

```
def get_required_info(ds_info, remove_missing=True):
   glosses = []
   gloss_video_id_map = {}
   signer_info = {}

   for word in ds_info:
      gloss = word['gloss']
      glosses.append(gloss)

      for inst in word['instances']:
         if (not os.path.exists(f'dataset/WLASL/videos/{inst["
            video_id"]}.mp4')) and remove_missing:
            continue

         if gloss not in gloss_video_id_map:
            gloss_video_id_map[gloss] = []

         gloss_video_id_map[gloss].append(inst['video_id'])

         s_id = inst['signer_id']

         if s_id not in signer_info:
            signer_info[s_id] = {}

         if gloss not in signer_info[s_id]:
            signer_info[s_id][gloss] = []

         signer_info[s_id][gloss].append(inst['video_id'])

      gloss_video_id_map[gloss].sort()

   return glosses, gloss_video_id_map, signer_info

with open('dataset/WLASL/info.json' , 'r') as ds_info_f:
   ds_info = json.load(ds_info_f)
```

```
glosses, gloss_video_id_map, signer_info = get_required_info(
   ds_info)
```

7.3 Preprocessing the Videos

```
from fractions import Fraction

def save_holistic(path, holistic, image):
   image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
   image.flags.writeable = False
   results = holistic.process(image)
   image.flags.writeable = True

   pose = np.array([[res.x, res.y, res.z, res.visibility] for
      res in results.pose_landmarks.landmark]).flatten() if
      results.pose_landmarks else np.zeros(33*4)
   face = np.array([[res.x, res.y, res.z] for res in results.
      face_landmarks.landmark]).flatten() if results.
      face_landmarks else np.zeros(468*3)
   lh = np.array([[res.x, res.y, res.z] for res in results.
      left_hand_landmarks.landmark]).flatten() if results.
      left_hand_landmarks else np.zeros(21*3)
   rh = np.array([[res.x, res.y, res.z] for res in results.
      right_hand_landmarks.landmark]).flatten() if results.
      right_hand_landmarks else np.zeros(21*3)

   landmarks = np.concatenate([pose, face, lh, rh])
   np.save(path, landmarks)
```

```
def sequential_frames(v_id, src, holistic, n_frames):
    for f in range(n_frames):
        ret, frame = src.read()
        if ret:
            save_holistic(f'./wlasl-info/landmarks-15-30/{v_id}/{f
                }.npy', holistic, frame)
        else:
            np.save(f'./wlasl-info/landmarks-15-30/{v_id}/{f}.npy'
                , np.zeros((1662, )))

def frames_from_video(v_id, n_frames):
    mp_drawing = mp.solutions.drawing_utils
    mp_drawing_styles = mp.solutions.drawing_styles
    mp_holistic = mp.solutions.holistic

    with mp_holistic.Holistic(min_detection_confidence=0.5,
        min_tracking_confidence=0.5) as holistic:
        src = cv2.VideoCapture(f'./dataset/WLASL/videos/{v_id}.
            mp4')
        video_length = src.get(cv2.CAP_PROP_FRAME_COUNT)
        os.makedirs(f'./wlasl-info/landmarks-15-30/{v_id}', mode
            =777)

        if video_length <= n_frames:
            frame_step = 0
            sequential_frames(v_id, src, holistic, n_frames)
        else:
            f = 0
            ratio = round(video_length/n_frames, 1)
            frac = Fraction(ratio).limit_denominator(10)

            if ratio == 1.7 or ratio == 1.9:
                frac = Fraction(ratio).limit_denominator(2)

            num, den = frac.numerator, frac.denominator
            while f < n_frames:
                for _ in range(min(den, n_frames-f)):
                    ret, frame = src.read()
                    if ret:
                        save_holistic(f'./wlasl-info/landmarks
                            -15-30/{v_id}/{f}.npy', holistic, frame)
                    else:
                        np.save(f'./wlasl-info/landmarks-15-30/{v_id
                            }/{f}.npy', np.zeros((1662, )))
                    f += 1
                for _ in range(num-den):
                    ret, frame = src.read()

        src.release()

for gloss in glosses_15:
    for v_id in gloss_video_id_map[gloss]:
        frames_from_video(v_id, 30)
```

7.4 One-Hot Encoding the Labels

```
glosses_15 = list(filter(lambda x: (len(gloss_video_id_map[x])
   > 21) and (len(gloss_video_id_map[x]) < 27), glosses))
y = [i for i in range(len(glosses_15))]
```

```
y = to_categorical(y).astype(int)
label_map = {label:y[num] for num, label in enumerate(
   glosses_15)}
```

7.5 Splitting the Dataset

```
def get_train_val_test_data(glosses, label_map, split_ratio
   =[0.8, 0.9]):

   result_train_v_ids = []
   result_val_v_ids = []
   result_test_v_ids = []
   result_train_labels = []
   result_val_labels = []
   result_test_labels = []

   for gloss in glosses:
      v_ids = gloss_video_id_map[gloss]
      split_1 = int(len(v_ids) * (split_ratio[0]))
      split_2 = int(len(v_ids) * (split_ratio[1]))
      result_train_v_ids.extend(v_ids[:split_1])
      result_val_v_ids.extend(v_ids[split_1:split_2])
      result_test_v_ids.extend(v_ids[split_2:])
      result_train_labels.extend([label_map[gloss]]*split_1)
      result_val_labels.extend([label_map[gloss]]*(split_2-
         split_1))
      result_test_labels.extend([label_map[gloss]]*(len(v_ids)
         - split_2))

   return np.array(result_train_v_ids), np.array(
      result_train_labels), np.array(result_val_v_ids), np.
      array(result_val_labels), np.array(result_test_v_ids), np
      .array(result_test_labels)

a, b, c, d, e, f_ = get_train_val_test_data(glosses_15,
   label_map)
```

7.6 Implementing Keras Sequence

```
class DataGenerator(tf.keras.utils.Sequence):
   def __init__(self, v_ids, labels, n_frames, batch_size = 32,
       training = False):
      self.v_ids = v_ids
      self.labels = labels
      self.n_frames = n_frames
      self.training = training
      self.batch_size = batch_size
      self.indexes = np.arange(len(self.v_ids))
      if self.training:
         np.random.shuffle(self.indexes)

   def get_video_frames(self, v_ids, classes):
      result_X = []
      result_y = []
      for index, v_id in enumerate(v_ids):
         window = []
         for f in range(self.n_frames):
            window.append(np.load(f'./wlasl-info/landmarks
               -15-30/{v_id}/{f}.npy'))
         result_X.append(np.stack(window, axis=0))
         result_y.append(classes[index])

      return np.array(result_X), np.array(result_y)

   def __getitem__(self, index):
      indexes = self.indexes[index*self.batch_size:(index+1)*
         self.batch_size]
      return self.get_video_frames(self.v_ids[indexes], self.
         labels[indexes])

   def __len__(self):
      return math.floor(len(self.v_ids) / self.batch_size)

   def on_epoch_end(self):
      if self.training:
         np.random.shuffle(self.indexes)
train_gen = DataGenerator(a, b, 30, training=True)
val_gen = DataGenerator(c, d, 30, batch_size = 4)
```

7.7 Model Training

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, GRU

model = Sequential()

model.add(GRU(512, return_sequences=True, activation='relu',
   input_shape=(30, 1662)))
model.add(GRU(512, return_sequences=True, activation='relu',
   recurrent_dropout=0.4, dropout=0.4))
```

```
model.add(GRU(256, return_sequences=False, activation='relu',
   recurrent_dropout=0.6, dropout=0.6))
model.add(Dense(128, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(15, activation='softmax'))
model.compile(optimizer='adam', loss='categorical_crossentropy'
   , metrics=['accuracy'])

steps_per_epoch = len(train_gen)
validation_steps = len(val_gen)

history = model.fit(train_gen, epochs=500, steps_per_epoch=
   steps_per_epoch, validation_data=val_gen, validation_steps=
   validation_steps)
save_best_cb = keras.callbacks.ModelCheckpoint('wlasl-info/
   models/wlasl-15W', save_best_only=True)
history_3 = model.fit(train_gen, epochs=50, steps_per_epoch=
   steps_per_epoch, validation_data=val_gen, validation_steps=
   validation_steps, callbacks = [save_best_cb])
```

7.8 Testing the Best Model

```
best_model = keras.models.load_model('wlasl-info/models/wlasl
   -15W/')
test_gen = DataGenerator(e, f_, 30, batch_size=4)
best_model.evaluate(test_gen, steps=len(test_gen))
```

CHAPTER - 8
RESULTS AND PERFORMANCE ANALYSIS

CHAPTER - 8

RESULTS AND PERFORMANCE ANALYSIS

8 Results and Performace Analysis

Sign language recognition has become a popular research area in the field of computer vision and machine learning. The development of deep learning algorithms and the availability of large-scale sign language recognition datasets have contributed to significant progress in this area. In this project, we trained a deep learning model using a combination of pose, hand, and face landmarks extracted from videos of American Sign Language (ASL) gestures using the MediaPipe Holistic model. The model used a Gated Recurrent Unit (GRU) architecture to predict the corresponding word for the given ASL gesture. In this performance analysis, we evaluate the different metrics of the trained model on a test set and discuss the results. Furthermore, we analyze the impact of varying the number of frames used for training the model and investigate the performance of the model on different types of gestures.

8.1 Existing System

The C3D approach has been employed in the existing system for sign language gesture recognition, where two C3D models are trained on the hand and complete gesture regions respectively. The output from each model consists of 4096 features, which are then fused using a combination of Multi-Layer Perceptron (MLP) and autoencoder techniques. The MLP fusion technique yields an average accuracy of 85%, while the autoencoder fusion technique achieves an average accuracy of 83%. Overall, these are the results that the existing system demonstrates for sign language gesture recognition, particularly in the context of recognizing the gestures performed by people with varying sign language styles and backgrounds.

8.2 Proposed System

The proposed GRU model consists of three GRU layers with 512, 512, and 256 units respectively. The activation function used in the layers is ReLU. The first two layers have recurrent dropout and dropout of 0.4, while the last layer has a recurrent dropout and dropout of 0.6. The model also has three dense layers with 128, 64, and 32 units, respectively, and a final dense layer with 15 units and softmax activation function. The model takes as input a sequence of 30 frames from a video and outputs the predicted sign language word. The GRU layers in the model allow it to capture the temporal dependencies in the input sequence, while the dense layers help to reduce the dimensionality of the output and enable the final prediction.

8.2.1 Accuracy

Accuracy is a metric used to evaluate the performance of a model in machine learning. It measures the percentage of correct predictions made by the model on the total number of predictions. In other words, accuracy indicates how well a model can classify data points into the correct categories. It is calculated by dividing the number of correct predictions by the total number of predictions and multiplying the result by 100 to get a percentage. Accuracy can be affected by various factors, such as the quality of the data, the complexity of the model, and the size of the dataset. A higher accuracy indicates that the model is performing better, while a lower accuracy suggests that the model needs improvement. Equation 5 shows the formula for calculating accuracy.

$$Accuracy = \frac{Number\ of\ Correct\ Predictions}{Total\ Number\ of\ Predictions} \times 100 \quad (5)$$

where the number of correct predictions refers to the number of instances that the model predicted the correct output, and the total number of predictions is the total number of instances in the dataset that the model made predictions on. The accuracy is usually expressed as a percentage.
The GRU model has achieved an accuracy of 95% on the test data.

8.2.2 Precision

Precision is a metric used to evaluate the accuracy of a model's positive predictions. It measures the percentage of true positive predictions out of all positive predictions made by the model. In other words, precision indicates how well a model can identify the correct positive class. It is calculated by dividing the number of true positive predictions by the total number of positive predictions. Precision is important in situations where the cost of false positives is high, such as in medical diagnosis or fraud detection. A higher precision indicates that the model is making fewer false positive errors. Equation 6 shows the formula for calculating precision.

$$Precision = \frac{TP}{TP+FP} \times 100 \quad (6)$$

where, TP refers to the True Positives and FP refers to the False Positives
The GRU model has achieved an precision of 97% on the test set.

8.2.3 Recall

Recall is another metric used to evaluate the performance of a classification model. It measures the percentage of actual positive cases that are correctly identified by the model. In other words, recall tells us how many of the positive cases in the dataset were correctly identified as positive by the model. It is calculated by dividing the number of true positives (TP) by the sum of true positives and false negatives (FN), and multiplying the result by 100 to get a percentage. A high recall indicates that the model is good at identifying positive cases, while a low recall suggests that the model may be missing some positive cases.Equation 7 shows the formula for calculating recall.

The formula for recall is:

$$Recall = \frac{TP}{TP+FN} \times 100 \quad (7)$$

The GRU model has achieved an recall of 96% on the test set.

8.2.4 F1-Score

F1-score is a metric used to evaluate the overall performance of a classification model. It is calculated based on the precision and recall of the model's predictions. Precision is the proportion of true positive predictions out of all positive predictions, while recall is the proportion of true positive predictions out of all actual positives in the data. F1-score is the harmonic mean of precision and recall, which balances both metrics equally. It ranges from 0 to 1, with higher values indicating better performance. The formula for F1-score is shown in Equation 8.

$$F_1 - Score = 2 * \frac{Precision * Recall}{Precision + Recall} \quad \ldots\ldots\ldots\ldots \quad (8)$$

The GRU model has achieved an F1-Score of 0.95 on the test set.

8.2.5 Confusion Matrix

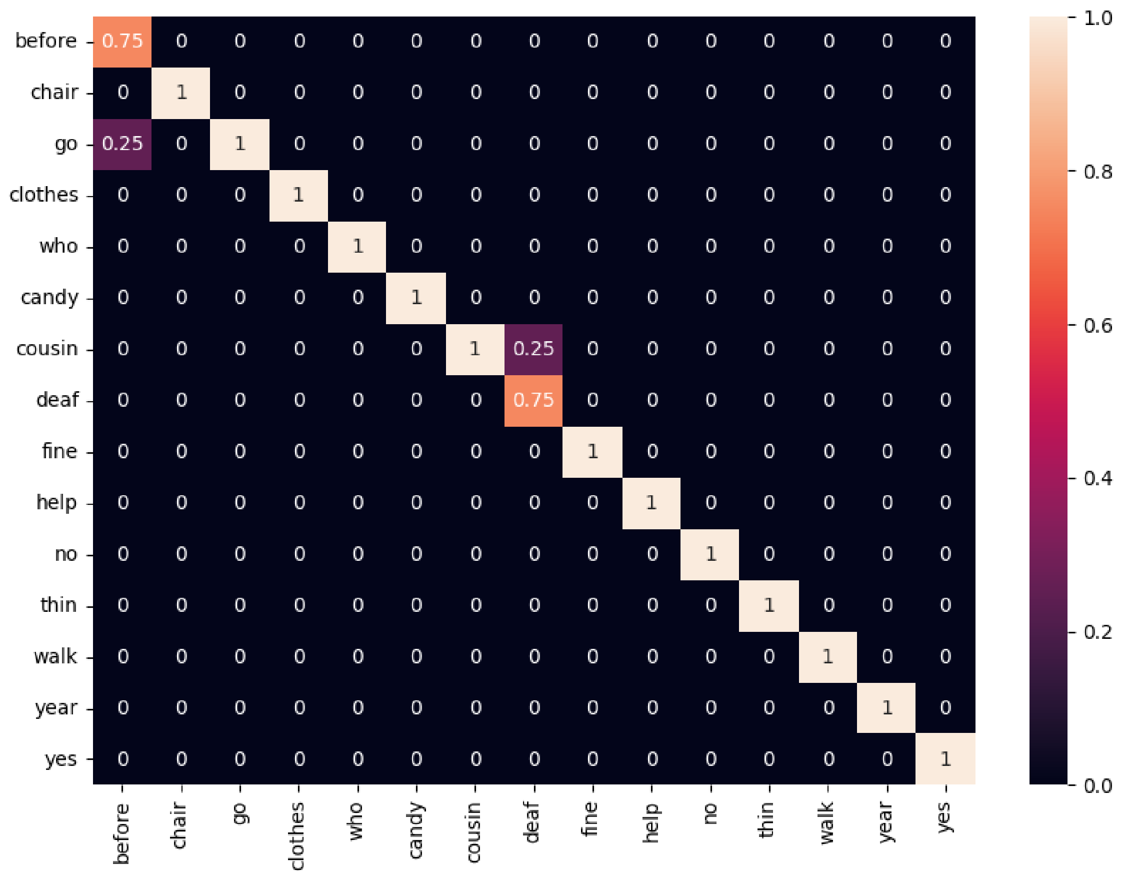


Fig 10: Confusion Matrix

A confusion matrix is a table that is used to evaluate the performance of a machine learning model. It shows the number of correct and incorrect predictions made by the model on a dataset, divided into various categories or classes. In a binary classification problem, the confusion matrix has two classes: positive and negative. The rows represent the actual class labels, while the columns represent the predicted class labels. The four outcomes that can be observed from a confusion matrix are true positive (TP), true negative (TN), false positive (FP), and false negative (FN). These outcomes are used to calculate various performance metrics such as accuracy, precision, recall, and F1 score. A confusion matrix helps to identify which classes the model is performing well on and which classes it needs to improve upon.

		Actual	
		Positive	Negative
Predicted	Positive	True Positive (TP)	False Positive (FP)
	Negative	False Negative (FN)	True Negative (TN)

The confusion matrix for the proposed model on 15 words is shown in Fig 10.

The 15 words are ‘before’, ‘chair’, ‘go’, ‘clothes’, ‘who’, ‘candy’, ‘cousin’, ‘deaf’, ‘fine’, ‘help’, ‘no’, ‘thin’, ‘walk’, ‘year’, ‘yes’.

8.2.6 ROC Curve

The ROC (Receiver Operating Characteristic) curve is a graphical representation used to evaluate the performance of a binary classification model. It is created by plotting the True Positive Rate (TPR) against the False Positive Rate (FPR) at various classification thresholds. The TPR is the proportion of actual positive cases that are correctly identified by the model, while the FPR is the proportion of actual negative cases that are incorrectly identified as positive by the model. The ROC curve helps in determining the optimal threshold for classification and can also be used to compare the performance of different models. A model with a higher ROC curve area under the curve (AUC) is considered better.

In different scenarios, the ROC curve can take different shapes. For example, if the ROC curve is a straight line passing through the origin, it suggests that the classifier is no better than random guessing. A perfect classifier would have a ROC curve that hugs the top left corner of the plot, indicating high TPR and low FPR across all threshold values.

A classifier with high specificity (low FPR) but low sensitivity (low TPR) would have a ROC curve that is close to the y-axis, while a classifier with high sensitivity (high TPR) but low specificity (high FPR) would have a ROC curve that is close to the x-axis.

The ROC curve for the GRU model is shown in Fig Fig 11. The GRU model can predict all the words fairly well but have a little bit of difficulty in predicting the word ’cousin’.

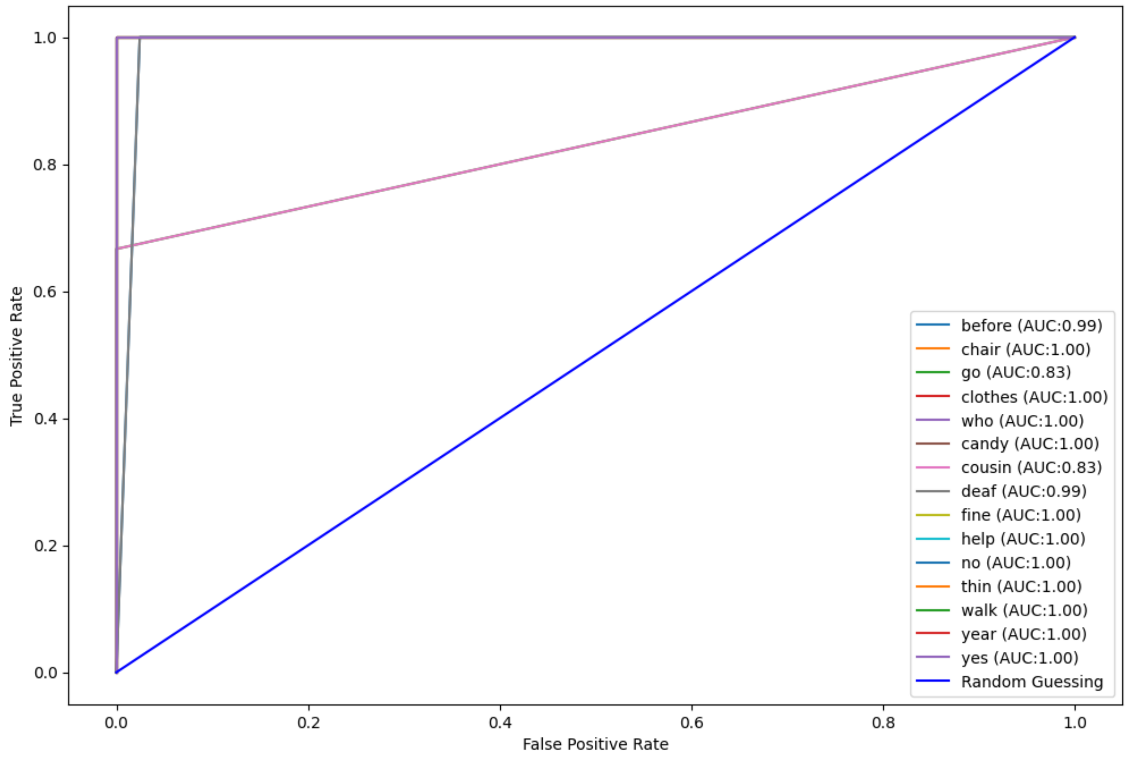


Fig 11: ROC Curve and AUC Score

8.2.7 Learning Curves

A learning curve is a figure that depicts the progression of a specific loss or evaluation metric related to learning during the training of a machine learning model. The x-axis represents time or progress, and the y-axis represents error or performance.

A learning curve can be used to determine whether or not a model is overfitting or underfitting. When a model is overfitting, it has a high training accuracy but a low validation accuracy. This means the model has memorized the training data and cannot generalize on new input. Conversely, when a model is underfitting, the training and validation accuracy will be low, indicating that the model cannot capture the patterns in the data.

The model is learning well as the learning curves shows a gradual decrease in both the training and validation loss, with both curves approaching each other and plateauing at similar low

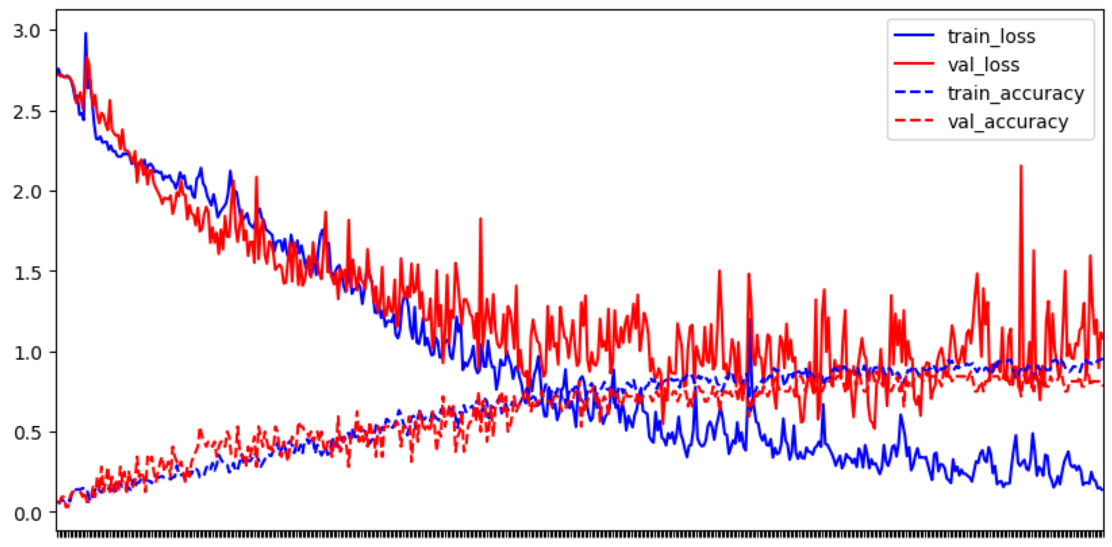


Fig 12: Learning curves for first 500 epochs

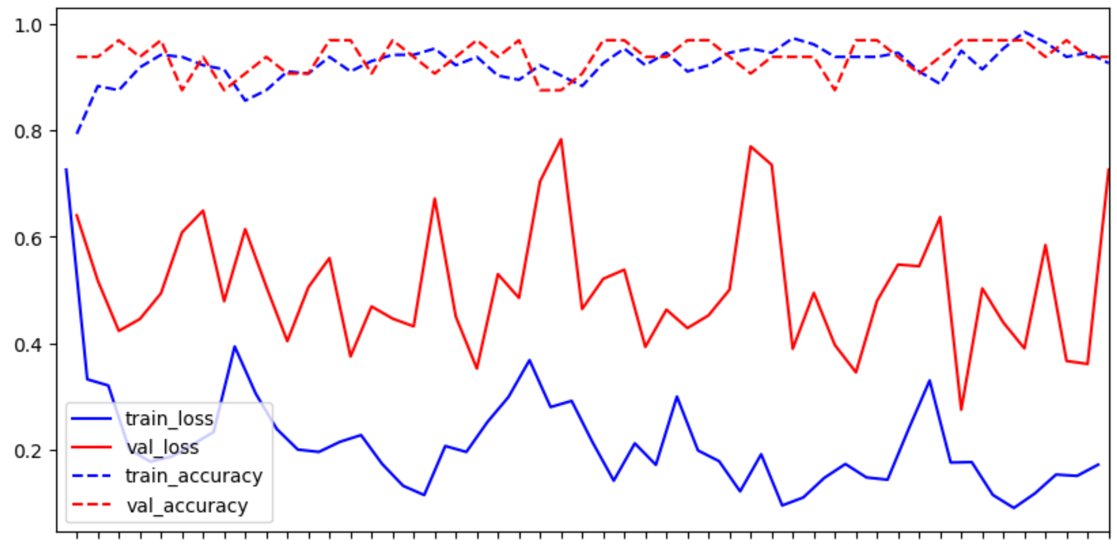


Fig 13: Learning curves for next 50 epochs

levels. This indicates that the model is performing well on both the training and validation data, and there is no significant gap between the two. The accuracy is also increased as the model is learning well which can be observed from the Fig 12 and Fig 13. The training and validation loss, with both curves approaching each other and plateauing at similar low levels, is an indicator that the model is learning well.

CHAPTER - 9

CONCLUSION AND FUTURE SCOPE

CHAPTER - 9

CONCLUSION AND FUTURE SCOPE

9 Conclusion and Future Scope

9.1 Conclusion

The above deep learning project was aimed at building a sign language recognition system using GRU models and MediaPipe Holistic Model. The project utilized the WLASL dataset, which is a large-scale sign language recognition dataset that contains videos of people performing sign language words.

The deep learning workflow involved systematic sampling of 30 frames from the video, retrieving pose, hand, and face landmarks from the MediaPipe Holistic Model, training a GRU model using the train dataset, and evaluating the model's performance on the test dataset.

The project's successful completion has demonstrated the potential of deep learning in developing accurate and efficient sign language recognition systems. The trained GRU model achieved an accuracy of 95% on the test dataset, which is a promising result.

Moreover, the project also explored the use of gTTS for generating speech, making the system more accessible to non-sign language users. Additionally, the project has showcased the potential of deep learning in addressing real-world problems and improving accessibility for people with disabilities.

9.2 Future Scope

1. One potential area for improvement is in expanding the dataset to include more diverse signers and gestures. This could improve the model's ability to generalize to different signers and variations in the same gesture.

2. Another area for improvement is in exploring the use of other deep learning architectures and techniques such as attention mechanisms and transformer networks. These techniques

have shown promising results in natural language processing tasks and may be applicable to sign language recognition as well.

3. Additionally, integrating real-time video processing can enhance the system's usability and accessibility.

References

[1] H. Luqman, “An efficient two-stream network for isolated sign language recognition using accumulative video motion,” *IEEE Access*, vol. 10, pp. 93785–93798, 2022.

[2] L. Pigou and N. Pugeault, “Towards sign language recognition with cnns: A benchmarking study,” 2019.

[3] M. Al-Qurishi, T. Khalid, and R. Souissi, “Deep learning for sign language recognition: Current techniques, benchmarks, and open issues,” *IEEE Access*, vol. 9, pp. 126917–126951, 2021.

[4] M. Al-Hammadi, G. Muhammad, W. Abdul, M. Alsulaiman, M. A. Bencherif, and M. A. Mekhtiche, “Hand gesture recognition for sign language using 3dcnn,” *IEEE Access*, vol. 8, pp. 79491–79509, 2020.

[5] M. Al-Hammadi and et al., “Deep learning-based approach for sign language gesture recognition with efficient hand gesture representation,” *IEEE Access*, vol. 8, pp. 192527–192542, 2020.

[6] N. Mohamed, M. B. Mustafa, and N. Jomhari, “A review of the hand gesture recognition system: Current progress and future directions,” *IEEE Access*, vol. 9, pp. 157422–157436, 2021.

[7] A. Mannan, A. Abbasi, A. R. Javed, A. Ahsan, T. R. Gadekallu, and Q. Xin, “Hypertuned deep convolutional neural network for sign language recognition,” *Computational Intelligence and Neuroscience*, vol. 2022, p. 1450822, 2022.

[8] B. Natarajan *et al.*, “Development of an end-to-end deep learning framework for sign language recognition, translation, and video generation,” *IEEE Access*, vol. 10, pp. 104358–104374, 2022.

[9] P. Chophuk and K. Chamnongthai, “Backhand-view-based continuous-signed-letter recognition using a rewound video sequence and the previous signed-letter information,” *IEEE Access*, vol. 9, pp. 40187–40197, 2021.

[10] Y. Liao, P. Xiong, W. Min, and J. Lu, “Dynamic sign language recognition based on video sequence with blstm-3d residual networks,” *IEEE Access*, vol. 7, pp. 38044–38054, 2019.

[11] A. Mittal, P. Kumar, P. P. Roy, R. Balasubramanian, and B. B. Chaudhuri, “A modified lstm model for continuous sign language recognition using leap motion,” *IEEE Sensors Journal*, vol. 19, no. 16, pp. 7056–7063, 2019.

[12] Z. Zhou, V. W. L. Tam, and E. Y. Lam, “Signbert: A bert-based deep learning framework for continuous sign language recognition,” *IEEE Access*, vol. 9, pp. 161669–161682, 2021.

[13] S. B. Abdullahi and K. Chamnongthai, “American sign language words recognition using spatio-temporal prosodic and angle features: A sequential learning approach,” *IEEE Access*, vol. 10, pp. 15911–15923, 2022.

[14] Q. M. Areeb, Maryam, M. Nadeem, R. Alroobaea, and F. Anwer, “Helping hearing-impaired in emergency situations: A deep learning-based approach,” *IEEE Access*, vol. 10, pp. 8502–8517, 2022.

[15] V. Nguyen, J. Starzyk, W. Goh, and D. Jachyra, “Neural network structure for spatio-temporal long-term memory,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 23, no. 6, pp. 971–983, 2012.

[16] M. Sincan and H. Keles, “Using motion history images with 3d convolutional networks in isolated sign language recognition,” *IEEE Access*, vol. 10, pp. 18608–18618, 2022.

[17] D. N. Kumar, M. Madhukar, A. Prabhakara, A. Marathe, and S. Bharadwaj, “Sign language to speech conversion — an assistive system for speech impaired,” in *2019 1st International Conference on Advanced Technologies in Intelligent Control, Environment, Computing & Communication Engineering (ICATIECE)*, pp. 272–275, IEEE, 2019.

[18] B. Joksimoski *et al.*, “Technological solutions for sign language recognition: A scoping review of research trends, challenges, and opportunities,” *IEEE Access*, vol. 10, pp. 40979–40998, 2022.

[19] W. Pan, X. Zhang, and Z. Ye, “Attention-based sign language recognition network utilizing keyframe sampling and skeletal features,” *IEEE Access*, vol. 8, pp. 215592–215602, 2020.

[20] I. Infantino, R. Rizzo, and S. Gaglio, “A framework for sign language sentence recognition by commonsense context,” *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 37, no. 5, pp. 1034–1039, 2007.