Sign to Text Conversion Using Deep Learning

Project report submitted to the Amrita Vishwa Vidyapeetham in partial fulfilment of the requirement for the Degree of

BACHELOR OF COMPUTER APPLICATIONS

Submitted by

Vishnu T R, Navaneeth R, Gopikrishna Rajmohan, Athul Krishna U and Pranav Mohan

(AM.SC.U3CDS21061, AM.SC.U3CDS21041, AM.SC.U3CDS21029, AM.SC.U3CDS21021 and AM.SC.U3CDS21046)

DEPARTMENT OF COMPUTER SCIENCE AND APPLICATIONS
AMRITA SCHOOL OF COMPUTING
AMRITA VISHWA VIDYAPEETHAM AMRITAPURI CAMPUS

JUNE 2024

DEPARTMENT OF COMPUTER SCIENCE AND APPLICATIONS

AMRITA VISHWA VIDYAPEETHAM AMRITAPURI CAMPUS

BONAFIDE CERTIFICATE

This is to certify that **Student Vishnu T R, Navaneeth R , Gopikrishna Rajmohan , Athul Krishna U and Pranav Mohan (AM.SC.U3CDS21061, AM.SC.U3CDS21041, AM.SC.U3CDS21029 ,AMSCU3CDS21021,AM.SC.U3CDS21046)**, a bonafide Bachelor of Computer Applications student at the Amrita School of Computing, Amrita Vishwa Vidyapeetham, has completed their undergraduate project titled *Your Project title here* under my guidance and supervision.

The project was completed successfully within the time frame allotted and the student has demonstrated a good understanding of the subject matter. The work carried out by the student is original and has not been submitted for any other academic purpose.

I certify that the student has fulfilled all the requirements necessary for the completion of the project and is eligible for the Bachelor of Computer Applications award. I wish all the success in their future endeavours.

Dr. Indulekha T S
Project Guide

Dr. Jinesh MK
Project Coordinator

Ms. Ani R
Chairperson
Dept. of CSA

Reviewer(s)

Place: Amritapuri Campus **Date:**

DEPARTMENT OF COMPUTER SCIENCE AND APPLICATIONS

AMRITA VISHWA VIDYAPEETHAM
AMRITAPURI CAMPUS

DECLARATION

We, Vishnu T R**, Navaneeth R, Gopikrishna Rajmohan , Athul Krishna U and Pranav Mohan (AM.SC.U3CDS21061, AM.SC.U3CDS21041, AM.SC.U3CDS21029 , AM.SC.U3CDS21021 and AM.SC.U3CDS21046)** hereby declare that this project entitled Sign to Text Conversion Using Deep Learning is a record of the original work done by us under the guidance of Dr. Indulekha T S, Amrita School of Computing, Amrita Vishwa Vidyapeetham that this work has not formed the basis for any degree/diploma/associationship/fellowship or similar awards to any candidate in any university to the best of our knowledge.

Signature of Student(s)

Dr. Indulekha T S

Project Guide

Place : Amritapuri Campus
Date :

Acknowledgements

I would like to express my sincere gratitude to all those who have helped me in completing this project successfully. This project has given me the opportunity to enhance my skills and knowledge in the field of computer applications.

First and foremost, we would like to acknowledge Sri. Mata Amritanandamayi Devi our Chancellor and guiding light for giving us the knowledge and opportunity to undertake this project satisfactorily.

I want to thank my project guide, Dr.Indulekha T S, for her constant support and guidance throughout the project. Her valuable suggestions and feedback have been instrumental in shaping this project.

I am also grateful to Dr.Jinesh MK, Assistant Professor (Sr), for providing me with the necessary resources and infrastructure to carry out this project. I want to thank the management of [Organization Name] for allowing me to work on this project.

Last but not least, I would like to acknowledge the contributions of my fellow team members who have worked alongside me on this project. Their dedication and teamwork have been the key to the success of this project.

Vishnu TR
Navaneeth R
Gopikrishna Rajmohan
Athul Krishna U
Pranav Mohan

Abstract

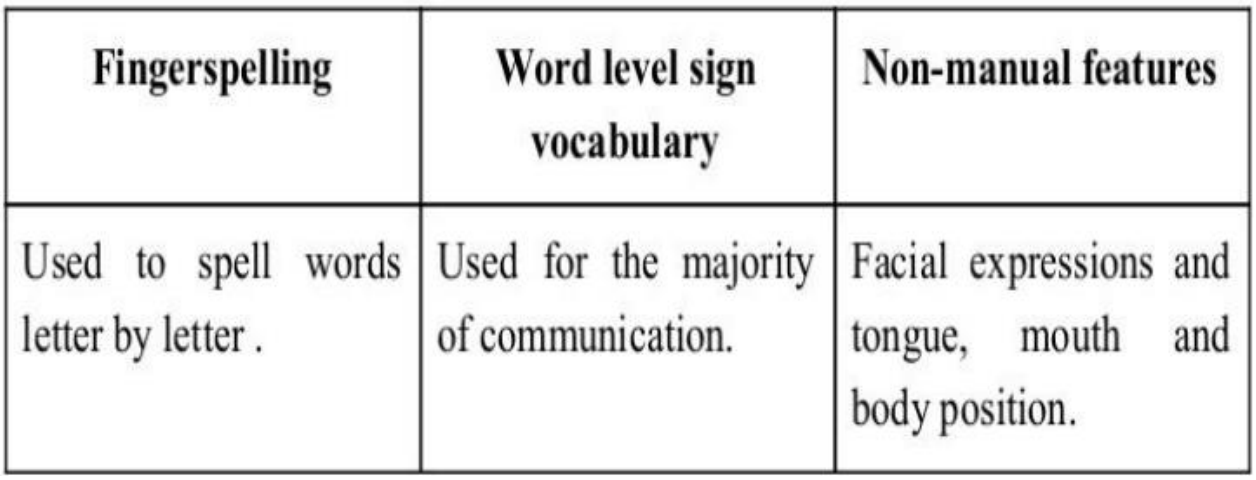
According to WHO(World health organization) approximately 70million people in the world are deaf-mute, This population faces a significant communication gap in interacting with people around them, Sign language is one of the reliable method to communicate with the deaf and mute, however vast majority of common people are unfamiliar with sign symbols and gestures, and to close this gap researches worldwide had been working on a technology that can convert these sign language gestures into text format. Automatic sign language recognition and conversion to text holds immense potential for bridging the communication gap between the deaf and hearing communities. This project explores a deep learning approach for sign language to text conversion using a combination of Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) networks. The proposed system leverages the power of CNNs in converting the ASL alphabets to text and LSTM is used to convert common gestures like hello, thanks, sorry, help into text format.

Contents

1. Introduction

Sign languages are complete and complex communication systems, distinct from spoken languages. There are hundreds of different sign languages used around the world, each with its own grammar, vocabulary, and syntax. These languages rely on a combination of hand shapes, facial expressions, and body movements to convey meaning. Chinese and Japanese, whose languages use the same body of characters but pronounce them entirely differently, can communicate by means of a sign language in which one watches while the other traces mutually understood characters in his or her palm. Evidence of long use of sign language to communicate around mutually unintelligible languages exists for Africa, Australia, and North America.

Sign language consists of 3 major components



Fingerspelling	Word level sign vocabulary	Non-manual features
Used to spell words letter by letter .	Used for the majority of communication.	Facial expressions and tongue, mouth and body position.

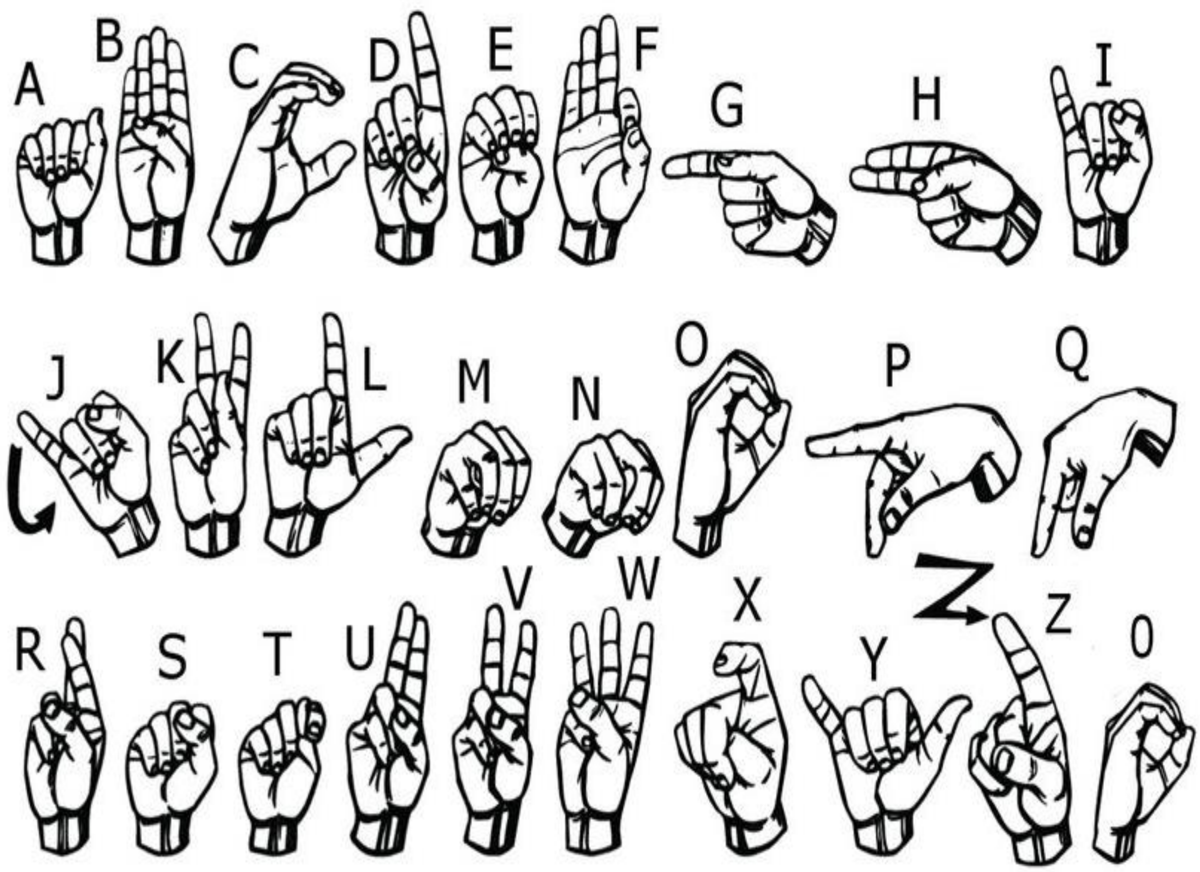
This project focuses on developing two independent sign-to-text translation models using deep learning techniques. The first model will be fingerspelling that utilize a Convolutional Neural Network (CNN) to recognize individual signs from the American Sign Language (ASL) alphabet. The second model will help us to translate non-manual features and will use a Long Short-Term Memory (LSTM) network to translate dynamic gestures like "hello," "thanks," "food," and "help" into text.The increasing demand for improved communication technologies between the deaf and hard-of-hearing community and the general population motivates this project. By creating separate models for static signs and dynamic gestures, we aim to:

- **Enhance ASL alphabet recognition:** The CNN model will tackle the challenge of accurately classifying individual ASL hand signs.
- **Develop gesture understanding:** The LSTM model will focus on recognizing and translating sequences of hand movements representing common gestures.
- **Promote communication accessibility:** Both models, working independently, can contribute to breaking down communication barriers.

Motivation: The growing need for accessible communication technologies between the deaf the following sections will delve deeper into the specifics of each model. We'll explore data preparation for ASL alphabets and gestures, the CNN architecture for sign recognition, the integration of Media Pipe with the LSTM architecture for gesture recognition, and strategies for training and evaluating both models

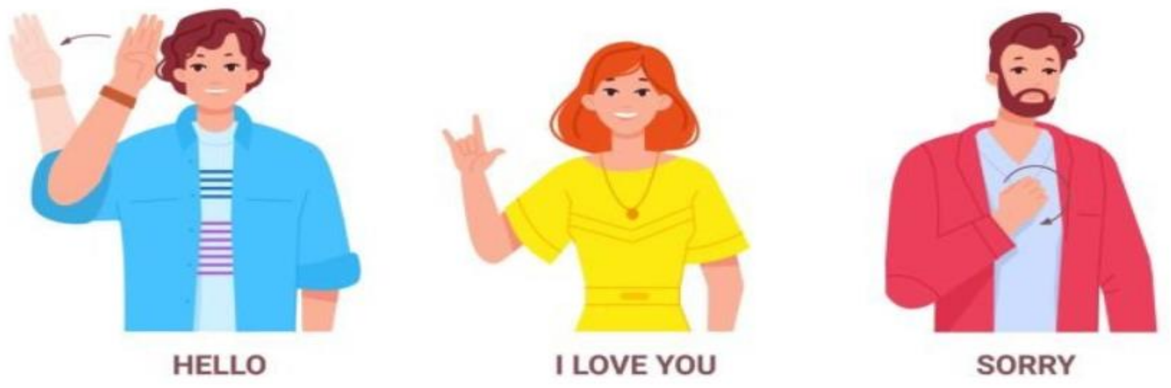
ASL Sign Language

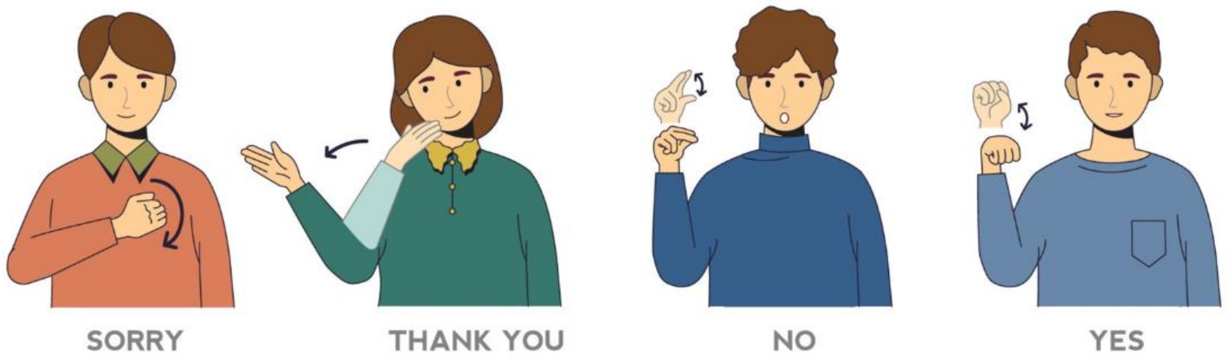
ASL (American Sign Language) is a complete, natural language used by the deaf and hard-of-hearing communities primarily in the United States and parts of Canada. ASL originated in the early 19th century with the establishment of the American School for the Deaf (ASD) in 1817 by Thomas Hopkins Gallaudet and Laurent Clerc. It incorporates elements from French Sign Language (LSF) and local sign languages. Over time, ASL evolved independently, developing its own unique grammar and lexicon. ASL has a distinct grammar, including its own rules for sentence structure, word order, and facial expressions, which serve as grammatical markers.



Common Gestures

sign languages, like ASL, are fantastic for communication, but individual handshapes (like those representing letters) can be tricky in real world scenarios like trying to tell a word by holding up letters one by one can be very difficult this is where gestures come.Gestures consist of a complete sign – the handshape, movement, and facial expression all working together. These gestures convey full ideas, not just letters. Saying "hello" by raising your open hand and giving a small shake This gesture is much easier to understand than holding the letter 'H' by itself. Focusing on complete gestures we can develop systems that translate sign language into spoken words,bridging communication gaps and empowering those who use sign language.





2. Problem Definition

The world of communication is vast, containing a vast of languages and methods of expression. However, a significant gap exists between spoken language and sign language, creating a barrier for the deaf and hard-of-hearing community. And through this project we seek to address this critical challenge by developing a real-time sign-to-text translation system that leverages the power of deep learning. The absence of a readily available sign language interpreter hinders spontaneous conversations between deaf and hearing individuals in various social settings. This can lead to feelings of isolation and exclusion for deaf individuals, impacting their ability to build relationships and participate fully in social activities. Also, Information often relies heavily on spoken language, leaving deaf and hard-of-hearing individuals at a disadvantage. This may limit their access to crucial information in educational settings, public services, healthcare and other areas. This can create a lot of Difficulty in comprehending announcements, lectures, or vital instructions which in-turn can significantly hinder their opportunities for learning, growth, and overall well-being.

The limitations imposed by the communication barrier have far-reaching consequences for the deaf and hard-of-hearing community. These consequences can impact various aspects of their lives:

Educational Obstacles: Difficulty accessing information in classrooms and limited opportunities for real-time interaction with teachers and peers can hinder academic performance and limit educational opportunities.

Employment Challenges: Communication difficulties during job interviews and challenges in understanding work-related instructions can pose significant barriers to employment and career advancement.

Reduced Independence: Even Simple tasks like shopping, navigating public transportation, or seeking medical attention can become more complex due to communication challenges. This can lead to increased dependence on others and a loss of autonomy.

So, there is a need for a Solution, and that solution is to create a correct, accessible, and user-friendly sign-to-text translation system that can be an effective tool in bridging the communication gap. And this project aims to develop such a system by leveraging deep learning techniques, specifically through CNN & RNN we will create 2 models one using CNN and one using RNN(LSTM).

ASL Alphabet Recognition: A CNN-based model will be designed to recognize individual ASL hand signs, translating them into corresponding text characters. This will address the need to understand static signs used in ASL communication.

Dynamic Gesture Recognition: An LSTM network, integrated with Media Pipe for hand pose estimation, this will translate dynamic gestures commonly used in sign language (e.g., "hello," "thanks," "food," "help") into text. This model will address the temporal aspects of sign language communication.

By creating these separate models, we aim to provide a more comprehensive solution that captures both the static and dynamic elements of ASL.
Developing a reliable sign-to-text translation system has the potential to significantly improve the lives of deaf and hard-of-hearing individuals. It can foster greater inclusion and accessibility by:

Facilitating Real-time Communication: This system can enable more spontaneous and direct communication between deaf and hearing individuals, promoting stronger social connections and a greater sense of belonging.

Enhanced Educational Opportunities: Improved access to information in classrooms will empower D/HH students to participate more actively in their education, leading to better academic outcomes.

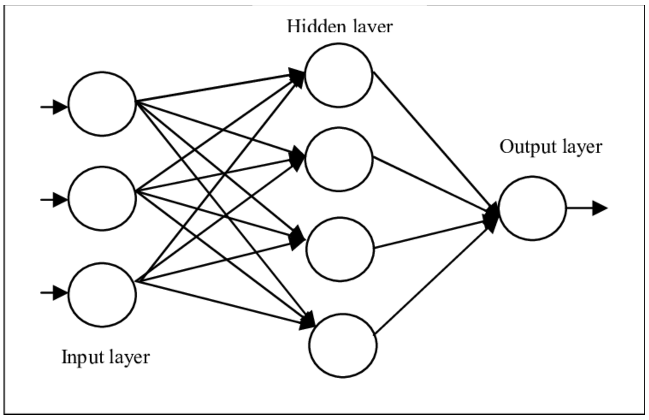
Increased Employment Prospects: By breaking down communication barriers in interviews and workplaces, the system can open doors to new employment opportunities and career advancement for D/HH individuals.

Greater Independence and Autonomy: The ability to communicate effectively will empower D/HH individuals to navigate daily tasks and social interactions with greater ease, fostering a sense of independence and self-reliance.

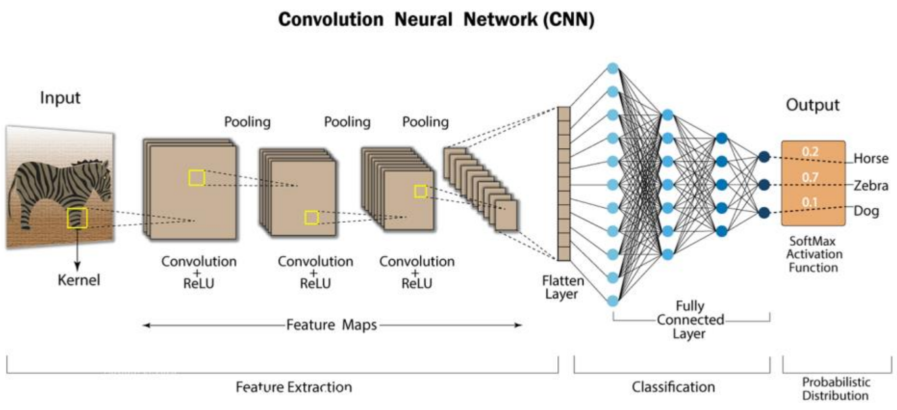
Improved communication capabilities can empower deaf and hard-of-hearing individuals to participate more actively in their communities, fostering a sense of inclusion and social connection.

3. Background

- **Artificial Neural Network (ANN)** is a connection of neurons, replicating the structure of human brain. Each connection of neuron transfers information to another neuron. Inputs are fed into first layer of neurons which processes it and transfers to another layer of neurons called as hidden layers. After processing of information through multiple layers of hidden layers, information is passed to final output layer.



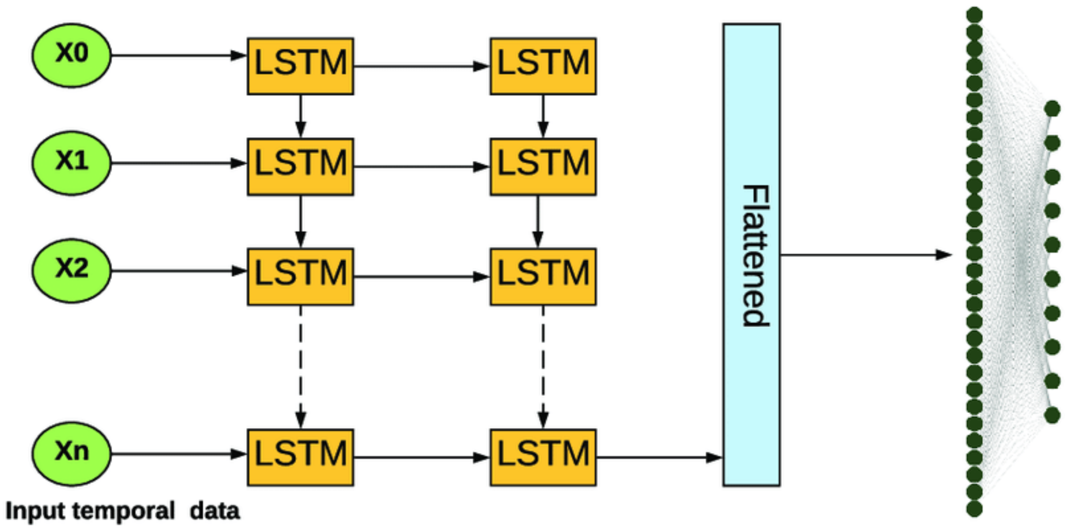
- **Convolutional Neural Networks (CNNs):** These are multi-layered artificial neural networks specifically designed for image recognition. CNNs utilize convolutional layers with filters that extract features like edges, shapes, and textures from images. Pooling layers then down sample these features, reducing the data size while retaining key information. Through repeated convolution and pooling, CNNs learn complex hierarchical representations of the input image, allowing them to effectively classify objects or handshapes in our case.



- Pooling Layer: We use pooling layer to decrease the size of activation matrix and ultimately reduce the learnable parameters. There are two types of pooling: a) Max Pooling: In max pooling we take a window size [for example window of size 2*2], and only take the maximum of 4 values. Well lid this window and continue this process, so well finally get an activation matrix half of its original Size. b) Average Pooling: In average pooling we take average of all values in a window.

- Fully Connected Layer: In convolution layer neurons are connected only to a local region, while in a fully connected region, well connect the all the inputs to neurons. 4. Final Output Layer: After getting values from fully connected layer, well connect them to final layer of neurons [having count equal to total number of classes], that will predict the probability of each image to be in different classes.

- **Long Short-Term Memory (LSTM) Networks:** LSTMs are a type of recurrent neural network (RNN) architecture specifically designed to handle sequential data like video frames. Unlike traditional RNNs that struggle with long-term dependencies, LSTMs have a unique architecture with memory cells that allow them to store information for extended periods. This enables LSTMs to capture the temporal dynamics of sign language gestures, where the order and sequence of hand movements are crucial for conveying meaning.



- **Sequential Model:** The Sequential model is a fundamental building block in Keras for constructing neural networks with a linear stack of layers. This approach is well-suited for our CNN and LSTM models, where each layer performs a specific task in the processing pipeline. For example, in the CNN model, the sequential layers might include convolutional layers for feature extraction, pooling layers for dimensionality reduction, and fully connected layers for classification of the ASL hand sign.

Essential Libraries:

- **Keras :** Keras is a high-level API that sits on top of deep learning frameworks like TensorFlow or PyTorch. It offers several advantages for building and training our CNN and LSTM models. Keras provides a user-friendly interface with the concept of a Sequential model. This allows us to define our models by stacking layers sequentially, making the code more readable and easier to maintain

- **NumPy:** This fundamental library provides efficient array manipulation capabilities, which are essential for working with large datasets and performing numerical computations within the deep learning models.
- **Scikit-learn:** This library offers various tools for data preprocessing, model selection, and evaluation, which can be helpful in data preparation and model training

- **TensorFlow/PyTorch: P**opular deep learning frameworks that will be used to build, train, and deploy the CNN and LSTM models. Both frameworks offer high-level APIs, pre-built functions, and optimization tools for efficient deep learning development.

- **OpenCV (cv2):** OpenCV (often referred to as cv2 in Python) is a powerful library for computer vision tasks. It provides functionalities for image and video processing, which will be crucial for pre-processing the ASL image or video data before feeding it to the CNN or LSTM models. This may involve tasks like resizing images, normalization, and potentially hand segmentation to focus on the hand region.

- **MediaPipe:** MediaPipe is an open-source framework developed by Google for building multi-modal machine learning pipelines. It offers pre-built functionalities for tasks like pose estimation, object detection, and hand landmark tracking.MediaPipe offers pre-built solutions (called "graphs") for various tasks, including hand landmark tracking. This pre-built solution will be directly applicable to our gesture recognition model.

4. Related Work

Sign language translation systems are pivotal for bridging the communication gap between the deaf-dumb community and those who are not proficient in sign language. Traditional methods, such as sign language gloves, although effective, are often prohibitively expensive and cumbersome. Recent advancements in machine learning and deep learning offer promising alternatives through video-based sign language recognition. Sign language recognition is a multi-faceted problem involving the identification of complex and continuous hand movements. Early approaches relied heavily on hardware, such as sensor-based gloves, which provided precise motion capture but were costly and impractical for everyday use. Effective machine learning models require large datasets for training. In the early stages, there was a scarcity of comprehensive sign language datasets. Many datasets lacked detailed annotations, making it challenging to train models accurately. Sign languages are rich in expression and involve complex hand movements, facial expressions, and body postures, Capturing and interpreting continuous hand movements and transitions between signs posed significant challenges. Sign languages often include non-manual features like facial expressions and head movements, which are difficult to capture and interpret with traditional methods. Before the advent of advanced deep learning techniques, image and video processing capabilities were limited, Traditional computer vision methods struggled to extract relevant features from images and video frames, particularly under varying lighting conditions and backgrounds. Achieving real-time processing was challenging due to the computational complexity of analysing video streams. Overcoming these challenges required advancements in both hardware and software. The shift towards vision-based systems leverages the ubiquity of cameras and advances in computer vision to interpret gestures without specialized equipment.Deep learning, particularly CNNs and RNNs, has significantly advanced the capabilities of sign language recognition systems. CNNs are adept at feature extraction from images, identifying patterns such as edges and textures that represent different hand shapes and positions. RNNs, particularly Long Short-Term Memory networks (LSTMs), excel in handling sequential data, making them suitable for interpreting the temporal dynamics of hand gestures in video sequences. CNNs are particularly effective in processing spatial information from video frames. Their layered architecture allows for the extraction of hierarchical features, from simple edges in the initial layers to more complex shapes in the deeper layers. Studies have shown that CNNs can achieve high accuracy in static gesture recognition by learning discriminative features from hand images. RNNs, and especially LSTMs, are designed to handle temporal dependencies and sequential information. In the context of sign language recognition, RNNs process sequences of video frames, capturing the motion and transitions between different signs. This temporal awareness is crucial for distinguishing between similar gestures that vary based on their movement patterns. The integration of CNNs and RNNs combines the strengths of both architectures, resulting in robust systems capable of high-accuracy sign language translation. The CNN component extracts spatial features from individual frames, which are then fed into the RNN component to capture the temporal dynamics. This hybrid approach has been shown to outperform models that use either CNNs or RNNs alone, achieving an estimated accuracy of 92.4% on dynamic hand gesture datasets. The potential applications of real-time sign language translation systems are vast. They can enable deaf-dumb individuals to participate more fully in educational and professional settings, such as giving presentations or joining video conferences. These systems can convert recognized gestures into text and speech using Text-To-Speech APIs, further enhancing communication. Future research should focus on improving the robustness and scalability of these systems. Challenges remain in handling variations in lighting, background, and signer appearance.

5. Methodology

In this section, the design and architecture of the proposed solution are described. It includes an overview of the system components, their interactions, and how they collectively address the problem defined earlier.

The proposed system will consist of two independent, deep learning-powered models, each tackling a specific aspect of sign language communication:

Model 1: ASL Alphabet Recognition - This model utilizes a Convolutional Neural Network (CNN) to recognize individual ASL hand signs from images or video frames.

Model 2: Gesture Recognition - This model employs a Long Short-Term Memory (LSTM) network in conjunction with MediaPipe for hand pose estimation to translate dynamic gestures into text.

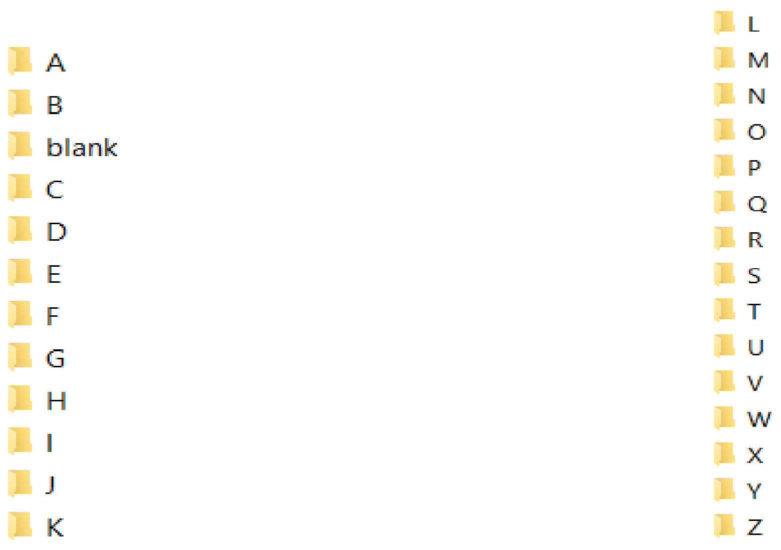
ASL Alphabet Recognition using CNN

We used the OpenCV (cv2) and operating system (os) libraries primarily for creating the dataset. Each letter is stored in a specific file named after the letter itself. We collected nearly 600 images for each alphabet, once the dataset was collected then it was spitted as training which is used to fit the model and validation was used to test the performance of the model
First, we capture each frame shown by the webcam of our machine in this frame we define a region of interest (ROI) which is denoted by a white square as shown in the image below

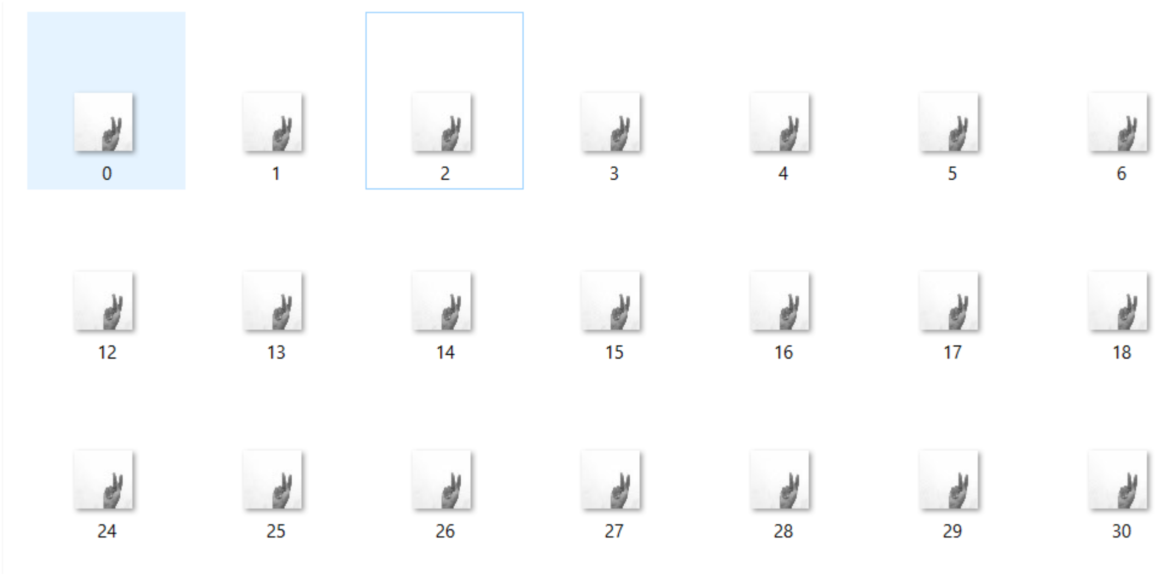
From this whole frame we extract only the ROI which is in RGB colour and convert it into a grayscale image, and then this grayscale image is resized into 48x48 pixel.

Dataset

SignImage48x48	04-05-2024 12:39	File folder
Splitdataset48x48	09-05-2024 23:08	File folder


A
B
blank
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
T
U
V
W
X
Y
Z

0 1 2 3 4 5 6 7 8 9
12 13 14 15 16 17 18 19 20 21
24 25 26 27 28 29 30 31 32 33
36 37 38 39 40 41 42 43 44 45


0 1 2 3 4 5 6
12 13 14 15 16 17 18
24 25 26 27 28 29 30

CNN

First Convolutional Layer

Conv2D (64, kernel size= (3,3), activation='relu'): The network starts with an input image of size **48x48 pixels** with one channel (grayscale). This layer applies 64 filters of size 3x3 to the input image. Each filter learns to detect specific features in the image. After convolution, the image might have a slightly reduced size due to the filter size and stride the output becomes **46x46 pixels** with **64 channels** (one for each filter).And then Maxpooling layer of size (2x2) comes This layer performs down sampling by taking the maximum value within each 2x2 window here the image size is reduced to 22X22 pixels while retaining essential features,after this drop out sets 20% of activations in the feature maps to zero during training to prevent overfitting. randomly sets

Second Convolutional layer

The input image to this layer will be of size 22x22, this layer uses 128 filters of size 3x3 there is a maxpooling layer after this where the image is further down sampled to 10x10 while maintaining 128 channels. MaxPooling**2D (pool size= (2,2))**: Another max pooling layer reduces the size. the drop layer which follows this will give final image of size 10x10 with 128 channels.

Third Convolutional layer

This layer introduces 512 filters of size 3x3.so the image will become 8x8 pixels with 512 channels and the final maxpooling layer scales down the image to 4x4 pixels.

Flatten Layer

The Flatten () layer transforms the multi-dimensional output (4x4x512 in our case) from the convolutional layers into a single, long vector 8192-dimensional vector, preparing it for input to the fully connected layers. This vector contains all the extracted features from the image, arranged in a linear fashion.

Dense Layers (Fully Connected)

Dense layers, also called fully connected layers, are the core of the classification process in a CNN. These layers connect each neuron in the previous layer to every neuron in the current layer, allowing for complex feature combination and classification.

Our model uses multiple dense layers with decreasing numbers of neurons (512, 256, 64) to progressively combine the extracted features from the convolutional layers and ReLU activation.

ReLU (Rectified Linear Unit) introduces non-linearity, allowing the network to learn more complex relationships between features. Dropout layers (with a 20% dropout rate) are used after each dense layer to prevent overfitting. They will randomly set activations to zero during training, encouraging the model to not rely on any specific feature too heavily.

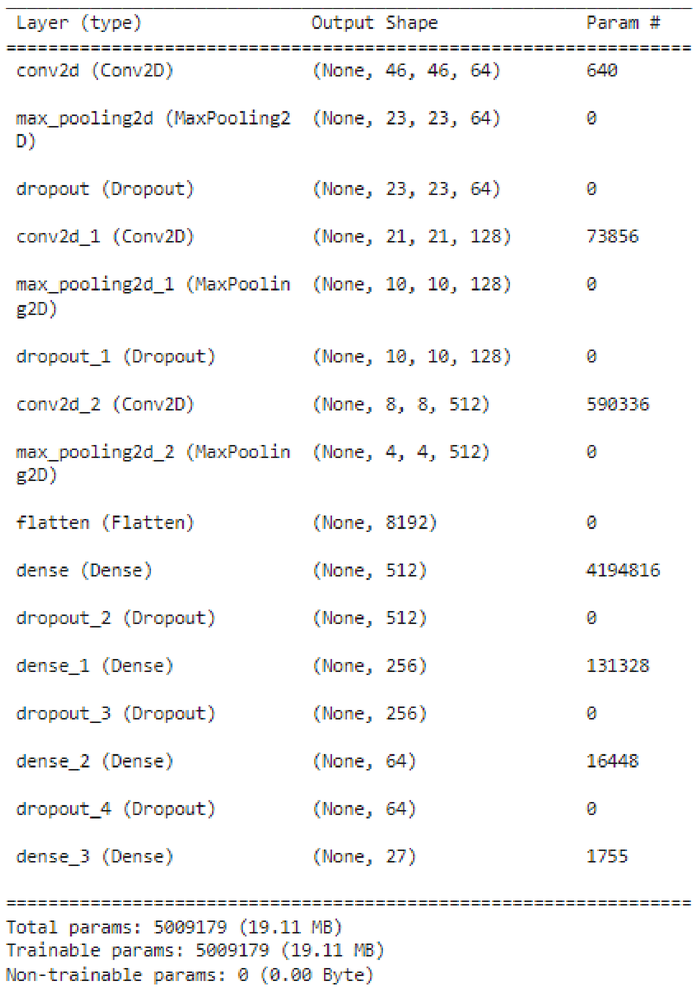
Final Dense Layer:

The final dense layer has 27 neurons corresponds to the number of classes that is 26 alphabets and one blank space that we are trying to classify. Here Each neuron learns to represent the probability of the input image belonging to a specific alphabet and uses SoftMax activation.

SoftMax Activation: SoftMax activation ensures the output of the final layer lies between 0 and 1, representing probabilities for each class. The class with the highest probability is predicted as the image's category

Fitting the model

As the dataset is too large it's fed into the model batchwise as each step, our model had 15 steps through which data of almost 10,000 images was trained. The epochs were set to 100. At last, the Model architecture is saved as Json and model weights are saved.



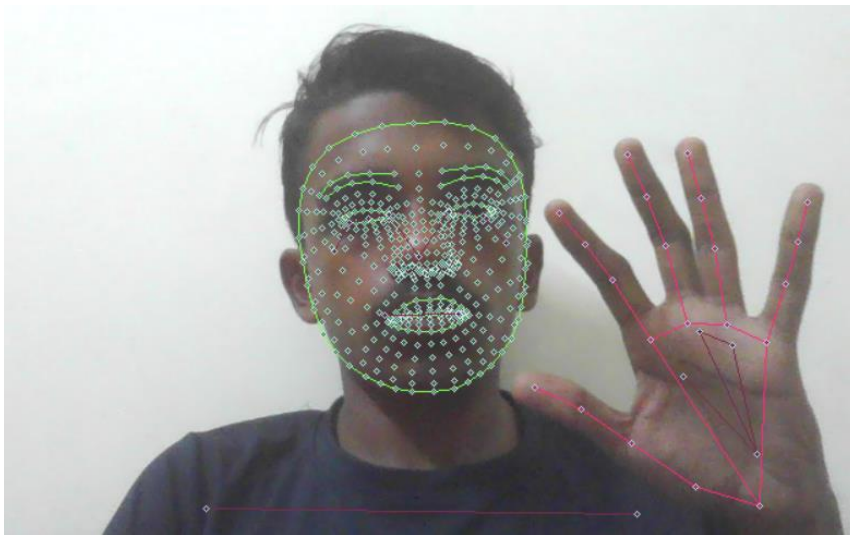
```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 46, 46, 64)        640

 max_pooling2d (MaxPooling2  (None, 23, 23, 64)        0
 D)

 dropout (Dropout)           (None, 23, 23, 64)        0

 conv2d_1 (Conv2D)           (None, 21, 21, 128)       73856

 max_pooling2d_1 (MaxPoolin  (None, 10, 10, 128)       0
 g2D)

 dropout_1 (Dropout)         (None, 10, 10, 128)       0

 conv2d_2 (Conv2D)           (None, 8, 8, 512)         590336

 max_pooling2d_2 (MaxPoolin  (None, 4, 4, 512)         0
 g2D)

 flatten (Flatten)           (None, 8192)              0

 dense (Dense)               (None, 512)               4194816

 dropout_2 (Dropout)         (None, 512)               0

 dense_1 (Dense)             (None, 256)               131328

 dropout_3 (Dropout)         (None, 256)               0

 dense_2 (Dense)             (None, 64)                16448

 dropout_4 (Dropout)         (None, 64)                0

 dense_3 (Dense)             (None, 27)                1755

=================================================================
Total params: 5009179 (19.11 MB)
Trainable params: 5009179 (19.11 MB)
Non-trainable params: 0 (0.00 Byte)
```

Realtime detection

We will import all necessary libraries for working, the saved model architecture is loaded this defines the structure of neural network, including number and type of layers used. If the architecture is loaded, then we load the trained weights these weights represent learned parameters. A webcam stream is captured using OpenCV's video capture class and in that a ROI (region of interest) is defined, a list named label is created which stores the character (A-Z), The ROI region is captured converted to grayscale and resized to 48X48 to ft the model requirements. The pre-processed image is fed into loaded CNN model which predicts the probabilities of each possible sign language(A-Z) in the label list, the predicted character with highest probability is defined using argmax function and its corresponding confidence score is displayed in the output frame

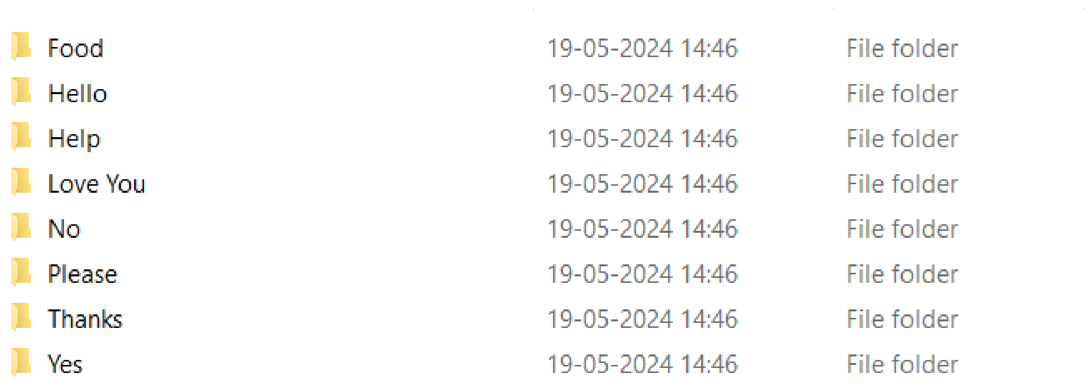
Gesture Recognition using LSTM

We use a combination of media pipe and lstm for gesture classification, we import holistic module from mediapipe library this helps us to detect human pose landmarks, face landmarks and hand landmarks simultaneously, along with holistic a drawing module (mp.solutions.drawing_utils) is defined this provides function for visualizing the results generated by holistic. Drawing function and detection function are written using this module which is integrated with the Cv2 function.so once the frame is captured it detect landmarks for pose, face, and hands. The x, y, and z values associated with each landmark represent its position in 3D space relative to a defined origin within the image frame. These relative positions and connections between landmarks, the model can infer the overall body posture of the person in the frame. The function returns image itself and a data structure containing result of detected landmarks.



The results are passed to a function to extract the key points and each KeyPoint like pose, face, right hand and left hand. Suppose we take pose landmarks then this function iterates through each landmark and creates a NumPy array containing the x, y, z coordinates, and visibility information If there is no extracted data, then it creates a NumPy array of size then a NumPy array is created of size 33*4 () similar logic is applied for extracting key points of face, right hand and left hand.

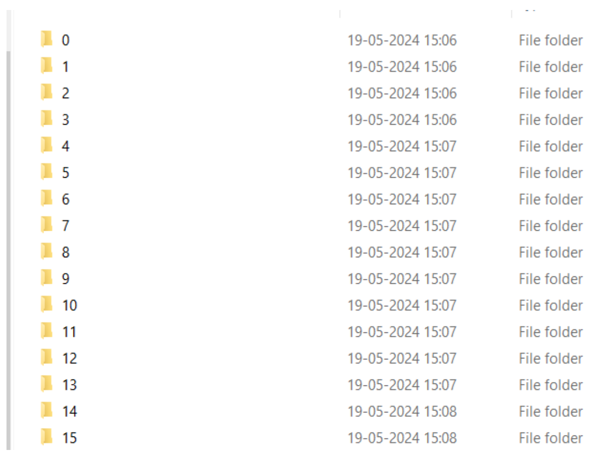
Then we will combine the extracted KeyPoint arrays for pose, face, left hand, and right hand into a single larger NumPy array. Now we will write a function to collect the KeyPoint for each gesture (‘Hello’,’Thanks’,’Love you’, ’yes’,’no’,’help’,’please’,’food’ these KeyPoint are used to train our model. We will create folders for each gesture to store the NumPy array.



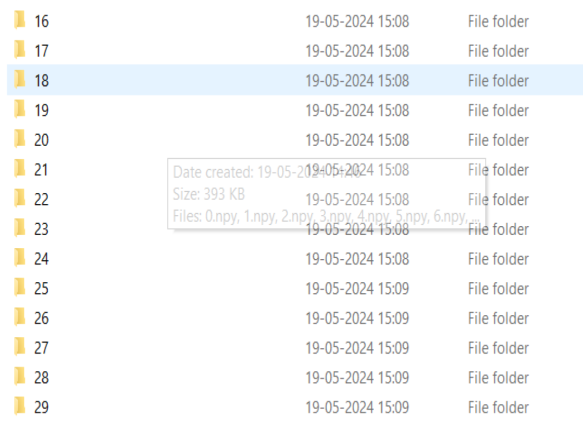
Food	19-05-2024 14:46	File folder
Hello	19-05-2024 14:46	File folder
Help	19-05-2024 14:46	File folder
Love You	19-05-2024 14:46	File folder
No	19-05-2024 14:46	File folder
Please	19-05-2024 14:46	File folder
Thanks	19-05-2024 14:46	File folder
Yes	19-05-2024 14:46	File folder

Next, we will write a function to collect datapoints for the gestures. The for loop will iterate through each action in the list. For each action we will capture 30 sequences and within each sequence we will collect 30 frames from the webcam.so each sequence will be like a short video of an action being performed. For each captured frame the holistic model will detect landmarks and store it as a NumPy array.

Sequence folder



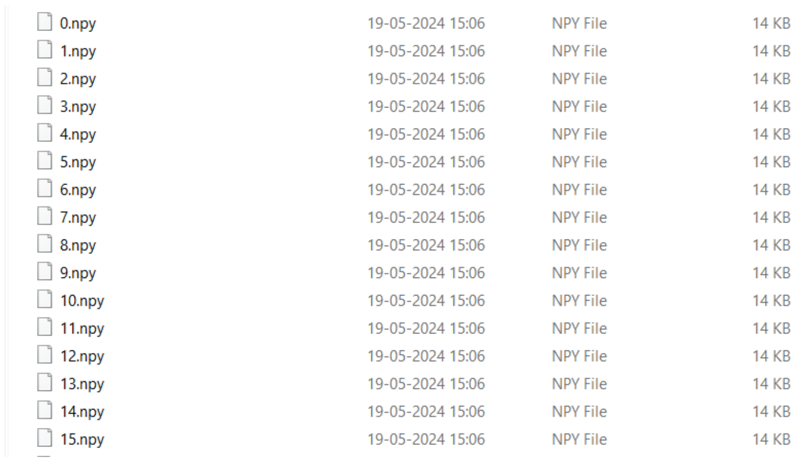
0	19-05-2024 15:06	File folder
1	19-05-2024 15:06	File folder
2	19-05-2024 15:06	File folder
3	19-05-2024 15:06	File folder
4	19-05-2024 15:07	File folder
5	19-05-2024 15:07	File folder
6	19-05-2024 15:07	File folder
7	19-05-2024 15:07	File folder
8	19-05-2024 15:07	File folder
9	19-05-2024 15:07	File folder
10	19-05-2024 15:07	File folder
11	19-05-2024 15:07	File folder
12	19-05-2024 15:07	File folder
13	19-05-2024 15:07	File folder
14	19-05-2024 15:08	File folder
15	19-05-2024 15:08	File folder



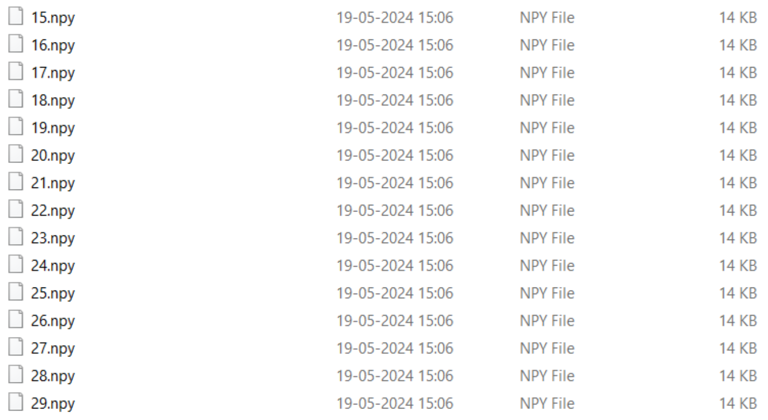
16	19-05-2024 15:08	File folder
17	19-05-2024 15:08	File folder
18	19-05-2024 15:08	File folder
19	19-05-2024 15:08	File folder
20	19-05-2024 15:08	File folder
21	19-05-2024 15:08	File folder
22	19-05-2024 15:08	File folder
23	19-05-2024 15:08	File folder
24	19-05-2024 15:08	File folder
25	19-05-2024 15:09	File folder
26	19-05-2024 15:09	File folder
27	19-05-2024 15:09	File folder
28	19-05-2024 15:09	File folder
29	19-05-2024 15:09	File folder

Date created: 19-05-20
Size: 393 KB
Files: 0.npy, 1.npy, 2.npy, 3.npy, 4.npy, 5.npy, 6.npy,...

Frame folder



0.npy	19-05-2024 15:06	NPY File	14 KB
1.npy	19-05-2024 15:06	NPY File	14 KB
2.npy	19-05-2024 15:06	NPY File	14 KB
3.npy	19-05-2024 15:06	NPY File	14 KB
4.npy	19-05-2024 15:06	NPY File	14 KB
5.npy	19-05-2024 15:06	NPY File	14 KB
6.npy	19-05-2024 15:06	NPY File	14 KB
7.npy	19-05-2024 15:06	NPY File	14 KB
8.npy	19-05-2024 15:06	NPY File	14 KB
9.npy	19-05-2024 15:06	NPY File	14 KB
10.npy	19-05-2024 15:06	NPY File	14 KB
11.npy	19-05-2024 15:06	NPY File	14 KB
12.npy	19-05-2024 15:06	NPY File	14 KB
13.npy	19-05-2024 15:06	NPY File	14 KB
14.npy	19-05-2024 15:06	NPY File	14 KB
15.npy	19-05-2024 15:06	NPY File	14 KB



15.npy	19-05-2024 15:06	NPY File	14 KB
16.npy	19-05-2024 15:06	NPY File	14 KB
17.npy	19-05-2024 15:06	NPY File	14 KB
18.npy	19-05-2024 15:06	NPY File	14 KB
19.npy	19-05-2024 15:06	NPY File	14 KB
20.npy	19-05-2024 15:06	NPY File	14 KB
21.npy	19-05-2024 15:06	NPY File	14 KB
22.npy	19-05-2024 15:06	NPY File	14 KB
23.npy	19-05-2024 15:06	NPY File	14 KB
24.npy	19-05-2024 15:06	NPY File	14 KB
25.npy	19-05-2024 15:06	NPY File	14 KB
26.npy	19-05-2024 15:06	NPY File	14 KB
27.npy	19-05-2024 15:06	NPY File	14 KB
28.npy	19-05-2024 15:06	NPY File	14 KB
29.npy	19-05-2024 15:06	NPY File	14 KB

This NumPy arrays (individual frame key points) for each sequence is transformed into a single sequence element within the sequences list. Each sequence element is a list containing the KeyPoint data for all frames within that sequence. A labels list will be there which holds the corresponding numerical label associated with the action performed in each sequence. This format (sequences and labels) is normally used for training machine learning models, particularly recurrent neural networks (RNNs) that process sequential data like our sign language detection.

Model Training
Sequence data is stored as X and label variable is stored as y then it is split into training set and test set which is used for model training.

LSTM Layer 1
This is the first layer of model its input shape is 30(1 sequence) each element is a vector of 1662 dimensions. The first layer has 64 units which aids the complexity of the layer and its capacity to learn patterns within the data.

There is a return sequence which is set into 'True' this instructs the LSTM layer to not only generate a final representation of the input sequence but also provide intermediate representations at each step within the sequence. This allows the model to capture long-term dependencies between KeyPoint across different frames in the sequence.

LSTM layer 2
The second LSTM layer has 128 units. This increased number of units compared to the first layer helps the model might be aiming to learn more complex patterns and relationships within the longer sequences. Like the first LSTM layer, return sequence is set to `True`. The activation function too is similar as first layer, ensuring non-linearity within the layer.

LSTM Layer 3

This is the final LSTM layer, again using 64 units. Here, the focus is to summarize the learned information from the previous layers into a concise representation of the entire sequence. So, unlike the first two, return sequences are set to False.

Dense Layer-1

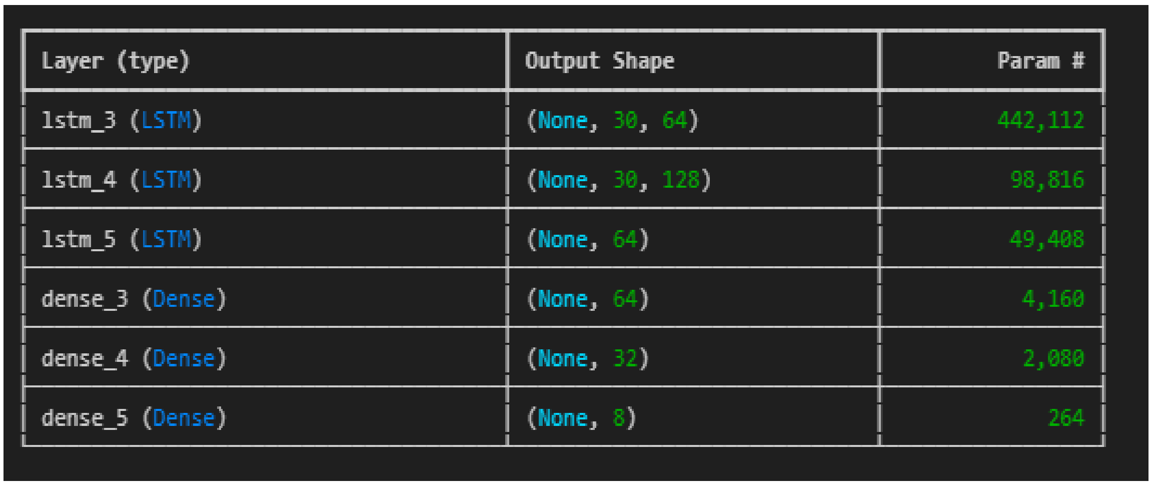
This is the first fully connected (dense) layer with 64 units. Dense layers transform the learned representation from the final LSTM layer into a format suitable for classification. The relu activation function is used for introducing non-linearity within the dense layer.

Dense Layer-2

This is the second dense layer with 32 units. It further processes the output from the previous dense layer, potentially refining the representation for classification.

Dense Layer-3

This is the final output layer. The number of units in this layer is set to the number of sign language actions in our dataset. Actions. Shape is ensuring the model can predict the class (action) from a set of actions stored. The SoftMax activation function is used here as its crucial for multi-class classification. It transforms the output from previous layer to a probability distribution across all possible action.



Layer (type)	Output Shape	Param #
lstm_3 (LSTM)	(None, 30, 64)	442,112
lstm_4 (LSTM)	(None, 30, 128)	98,816
lstm_5 (LSTM)	(None, 64)	49,408
dense_3 (Dense)	(None, 64)	4,160
dense_4 (Dense)	(None, 32)	2,080
dense_5 (Dense)	(None, 8)	264

The model is then compiled, during compiling Adam optimizer was used this is an efficient optimization algorithm. The model is fitted using X train, X test, y train, & y test we have set the epochs as 1000.

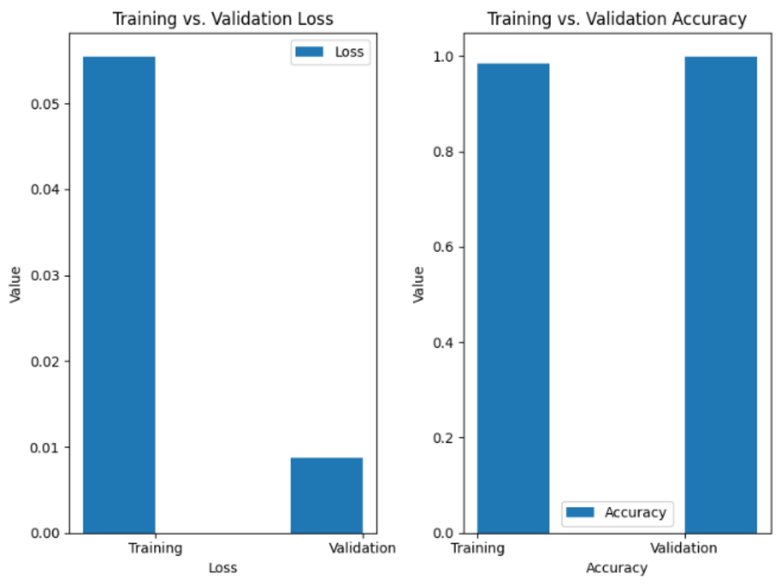
Realtime detection.

We will set an empty list to store the sequence, as the webcam is on it continuously captures 30 frames, and as said before for each frame KeyPoint data is extracted, once the sequence reaches 30 frames the model will analyse it to predict the most probable sign action that we have performed. Along with that we use coloured bar graphs to represent model's confidence level of different signs on the screen and for that we will iterate through the predicted probabilities and action list simultaneously this allows access to both probability value and corresponding action name.

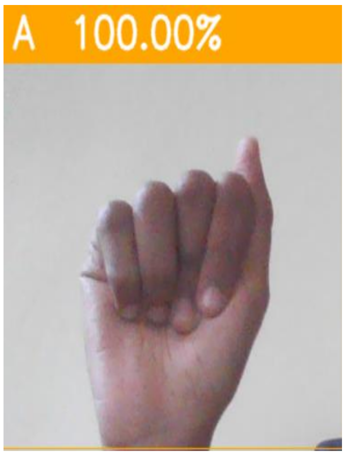
6. Result and Analysis

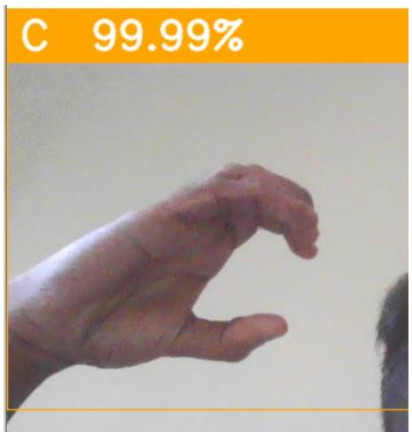
CNN

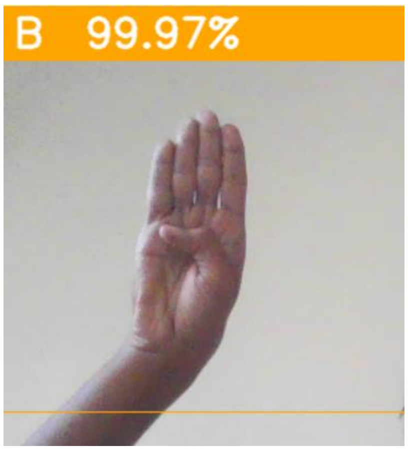
The model achieved a final training accuracy of **98.40%**, thus indicating a good fit on the training data. This means that on the training set, the model correctly classified **98.40%** of the sign language gestures. We have used the Categorical-cross entropy as the loss function the training loss of was almost 0.0554 average across all samples in batch or epoch suggesting model learned the patterns in the training data effectively. The model's performance on unseen data, evaluated using the validation set, is even more promising. It achieved a validation accuracy of **99%**, demonstrating excellent generalization capabilities. This signifies that our model can accurately classify sign language gestures it has not encountered during training. The validation loss was **0.0087** which further reinforces that the model is not overfitting to the training data and can perform well on new examples.

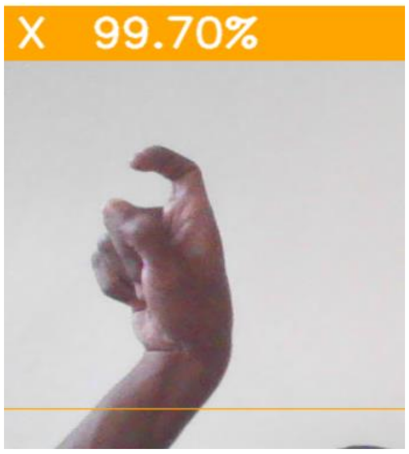


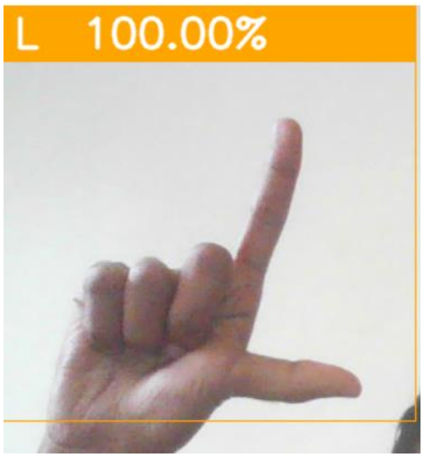
Real-Time Predictions

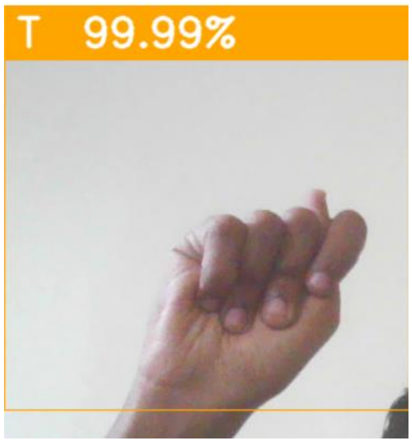












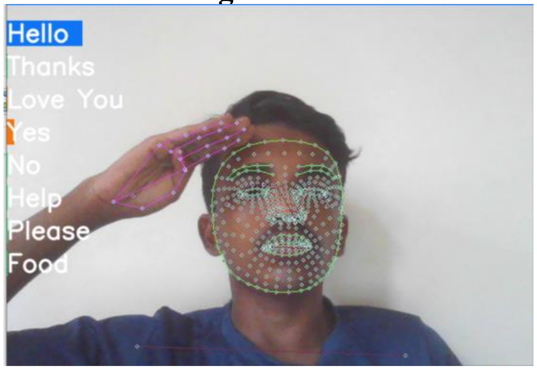
Our testing phase revealed valuable insights into the system's current performance:

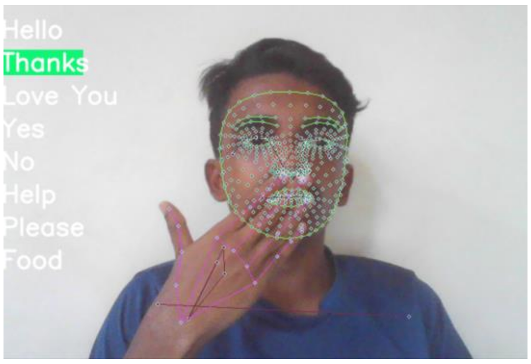
- **Lighting and Background Conditions:** Good lighting and a clear background significantly improved model accuracy. We will explore techniques to enhance robustness against variations in these factors.
- **Similar Signs:** The model sometimes confuses visually similar signs like 'A' and 'T,' 'D' with 'U,' 'R,' or 'V,' and 'U' with 'V' or 'W.' This arises due to very similar hand orientations.

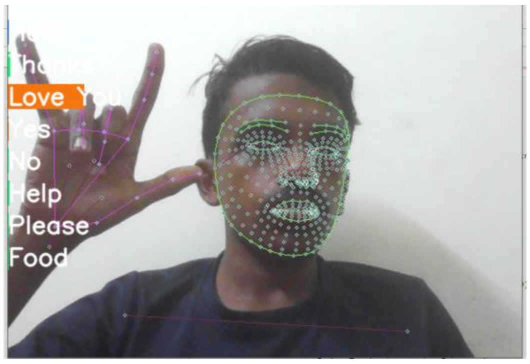
LSTM

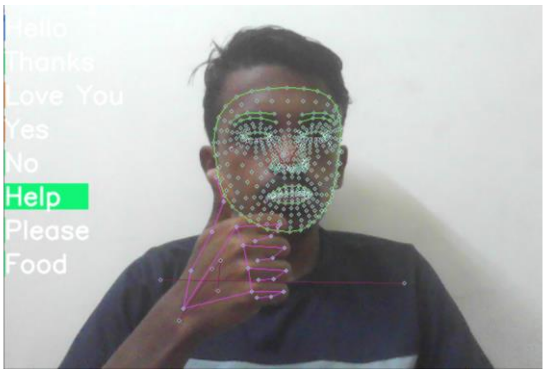
The LSTM model achieved a final training accuracy of **95.87%, indicating** a good fit on the training data. This means that on the training set, the model correctly classified **98.40%** of the sign language gestures. We have used the Categorical-cross entropy as the loss function the training loss of was 0.1260 average across all epochs suggesting model learned the patterns in the training data effectively. The model's performance on unseen data that is test data, achieved accuracy of **95%, whereas** the loss was 0.2704. This signifies our model is not overfitting on training data and can perform good on unseen or new data. The overall accuracy of the model was recorded as 95%.

Real-time testing









The testing phase of our LSTM-based gesture recognition model served as an invaluable window into its capabilities and limitations. The LSTM performed satisfactorily under good lighting, clear backgrounds, and slow, deliberate gestures, also the model takes some time to analyse the gesture and provide a confident prediction.so once we are showing a gesture, we should keep it for a while, The model's struggle with gestures that share similar hand positions at certain points, like "yes" & "food" or "thanks" & "sorry," highlights the inherent complexity of sign language recognition.

So, our probability graph rather than text helps to tackle these issues up to an extent, in certain situations. This graph depicts the model's confidence level for different gestures over time, providing valuable insights into its decision-making process. In cases where the model is unsure about the intended gesture, the user can potentially analyse the probability graph and make a more informed decision based on the peaks in the different gesture probabilities.

In Conclusion, the subtle changes in lighting and background introduced in such scenarios can significantly impact the model's ability to accurately interpret the gestures. Similarly, faster gestures, which are natural in everyday communication, can pose a challenge for the model's temporal processing capabilities.

7. Conclusion and Future Works

CNN-based Alphabet Recognition.

Data Augmentation for Increased Generalizability:

Exploring advanced data augmentation techniques that goes beyond basic lighting and background variations. This can include techniques like Random cropping, scaling, elastic deformations, and adding noise which can further enrich the training data and improve the model's ability to handle unseen variations in real-world scenarios.
We can also incorporate techniques like Generative Adversarial Networks (GANs) to create synthetic ASL hand sign images with even wider variations in lighting, background, clothing, and hand appearance. This can significantly increase the diversity of the training data. Along with these techniques collecting data in real-world with varying lighting background, user demographics & collecting images from multiple viewpoints can potentially improve model’s robustness.

2. Model Architectural Enhancements:

Experimenting with deeper CNN architectures that has more convolutional layers and pooling layers can potentially improve the model's ability to learn complex features from the hand sign images. Investigate architectures that are specifically designed for image classification tasks, such as VGGNet, Inception, or ResNet. These architectures have proven successful in various computer vision tasks and might offer advantages over the current CNN structure.

Ensemble Learning for Improved Robustness:

Train multiple CNN models with different architectures or hyperparameters and combine their predictions using techniques like averaging or weighted voting. This ensemble approach can lead to more robust and accurate predictions compared to a single model.

LSTM-based Gesture Recognition
The testing phase of the LSTM-based gesture recognition model served as a valuable experience, revealing us its strengths and limitations. While the model performs ok under controlled conditions, the real-world scenarios will pose some significant challenges.so we will delve into the potential future works which is aimed at enhancing accuracy, performance and user experience

Data Collection in Diverse Environments:
Moving beyond traditional controlled data collection and focusing data collection more on real world-environments will varying light conditions, backgrounds and user demographics Move beyond controlled lab settings and collecting data in real-world environments with varying lighting conditions, backgrounds, and user demographics addition incorporating data from various veiwpoints,which will give hand pose from multiple angles will potentially improve the model performance increasing its accuracy and effectiveness in real word. We could also explore techniques for adapting model for individual users by collecting a small amount of user-specific data during initial setup. This will fine-tune the model for that user's hand characteristics and signing style

Exploring Advanced LSTM Architectures:
Train multiple LSTM models with different architectures and hyperparameters then combine their predictions using techniques like averaging or weighted voting this will lead to more robust and accurate predictions and combine their predictions using techniques like averaging or weighted voting. This ensemble approach can lead to more robust and accurate predictions compared to a single model.

Hardware and Software Optimization:
Utilizing GPUs for faster model inference, enabling real-time gesture recognition.
Model Compression and Quantization:
Explore techniques like model compression or quantization that can reduce the model's size. This can be crucial for deploying the model on resource-constrained devices like smartphones.

Accessibility and Inclusivity:
Designing the system in a way such that it can be accessible for users with diverse abilities. This might involve incorporating alternative input modalities (e.g., eye tracking for users with limited hand mobility) and ensuring compatibility with assistive technologies.

Sign Language Generation:

Exploring the possibility of using deep learning techniques to generate sign language from spoken language or text input. This can have wide applications in education, accessibility, and machine translation.

Conclusion:

The development of an effective sign-text gesture model presents a significant challenge. However, the potential rewards are vast. By addressing the current limitations and actively exploring future research directions, there is for a future where technology bridges the communication gap and empowers individuals who rely on sign language.

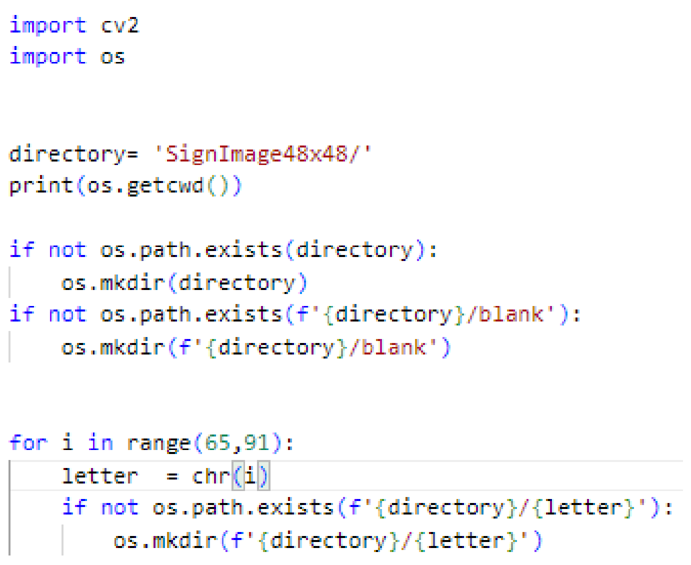
8. References

- Sonare, Babita, et al. "Video-based sign language translation system using machine learning." 2021 2nd International Conference for Emerging Technology (INCET). IEEE, 2021.
- Ananthanarayana, Tejaswini, et al. "Deep learning methods for sign language translation." ACM Transactions on Accessible Computing (TACCESS) 14.4 (2021): 1-30.
- Amin, Mohamed, Hesahm Hefny, and Mohammed Ammar. "Sign language gloss translation using deep learning models." International Journal of Advanced Computer Science and Applications 12.11 (2021).
- Kartik, P. V. S. M. S., et al. "Sign language to text conversion using deep learning." Inventive Communication and Computational Technologies: Proceedings of ICICCT 2020. Springer Singapore, 2021.

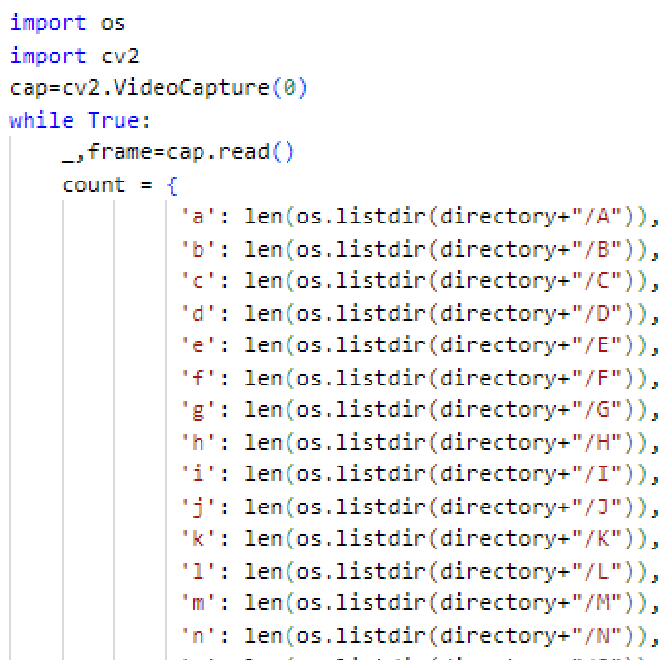
9. Source Code

CNN

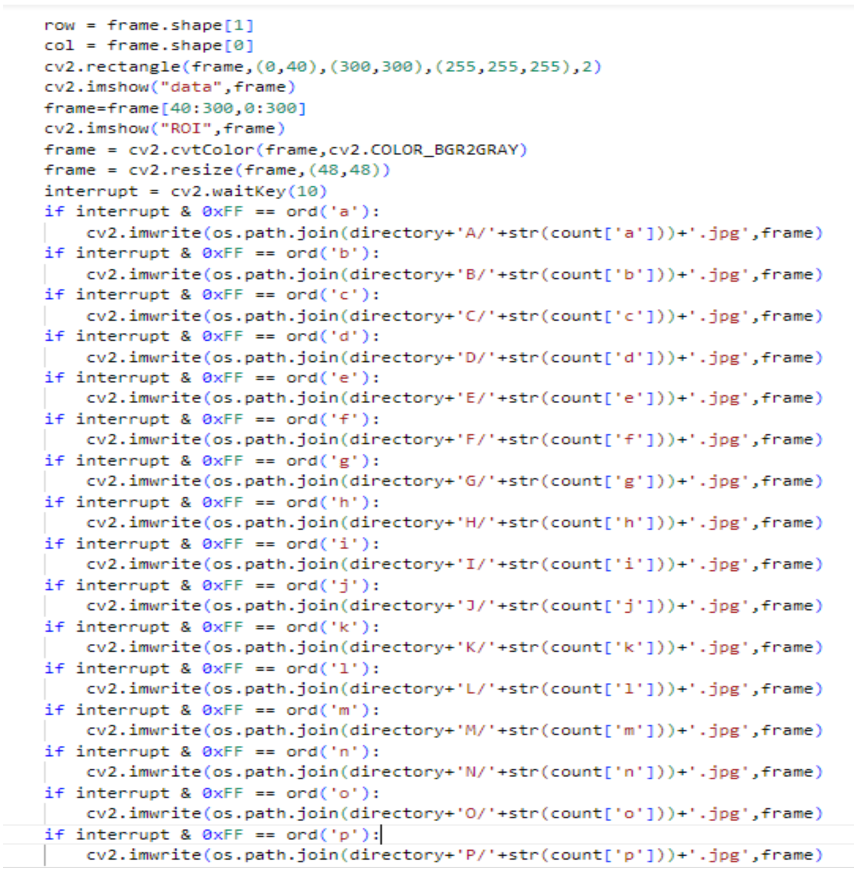
Dataset creation



```python
import cv2
import os


directory= 'SignImage48x48/'
print(os.getcwd())

if not os.path.exists(directory):
    os.mkdir(directory)
if not os.path.exists(f'{directory}/blank'):
    os.mkdir(f'{directory}/blank')


for i in range(65,91):
    letter  = chr(i)
    if not os.path.exists(f'{directory}/{letter}'):
        os.mkdir(f'{directory}/{letter}')
```

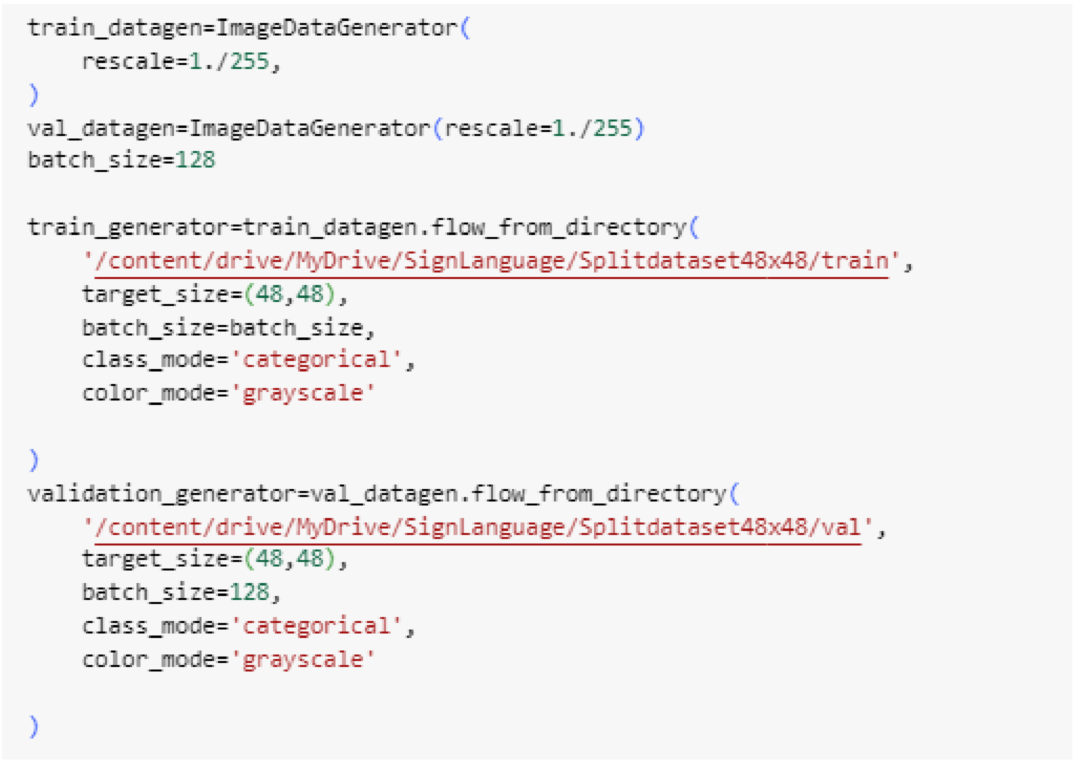


```python
import os
import cv2
cap=cv2.VideoCapture(0)
while True:
    _,frame=cap.read()
    count = {
                'a': len(os.listdir(directory+"/A")),
                'b': len(os.listdir(directory+"/B")),
                'c': len(os.listdir(directory+"/C")),
                'd': len(os.listdir(directory+"/D")),
                'e': len(os.listdir(directory+"/E")),
                'f': len(os.listdir(directory+"/F")),
                'g': len(os.listdir(directory+"/G")),
                'h': len(os.listdir(directory+"/H")),
                'i': len(os.listdir(directory+"/I")),
                'j': len(os.listdir(directory+"/J")),
                'k': len(os.listdir(directory+"/K")),
                'l': len(os.listdir(directory+"/L")),
                'm': len(os.listdir(directory+"/M")),
                'n': len(os.listdir(directory+"/N")),
```



```
row = frame.shape[1]
col = frame.shape[0]
cv2.rectangle(frame,(0,40),(300,300),(255,255,255),2)
cv2.imshow("data",frame)
frame=frame[40:300,0:300]
cv2.imshow("ROI",frame)
frame = cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)
frame = cv2.resize(frame,(48,48))
interrupt = cv2.waitKey(10)
if interrupt & 0xFF == ord('a'):
    cv2.imwrite(os.path.join(directory+'A/'+str(count['a']))+'.jpg',frame)
if interrupt & 0xFF == ord('b'):
    cv2.imwrite(os.path.join(directory+'B/'+str(count['b']))+'.jpg',frame)
if interrupt & 0xFF == ord('c'):
    cv2.imwrite(os.path.join(directory+'C/'+str(count['c']))+'.jpg',frame)
if interrupt & 0xFF == ord('d'):
    cv2.imwrite(os.path.join(directory+'D/'+str(count['d']))+'.jpg',frame)
if interrupt & 0xFF == ord('e'):
    cv2.imwrite(os.path.join(directory+'E/'+str(count['e']))+'.jpg',frame)
if interrupt & 0xFF == ord('f'):
    cv2.imwrite(os.path.join(directory+'F/'+str(count['f']))+'.jpg',frame)
if interrupt & 0xFF == ord('g'):
    cv2.imwrite(os.path.join(directory+'G/'+str(count['g']))+'.jpg',frame)
if interrupt & 0xFF == ord('h'):
    cv2.imwrite(os.path.join(directory+'H/'+str(count['h']))+'.jpg',frame)
if interrupt & 0xFF == ord('i'):
    cv2.imwrite(os.path.join(directory+'I/'+str(count['i']))+'.jpg',frame)
if interrupt & 0xFF == ord('j'):
    cv2.imwrite(os.path.join(directory+'J/'+str(count['j']))+'.jpg',frame)
if interrupt & 0xFF == ord('k'):
    cv2.imwrite(os.path.join(directory+'K/'+str(count['k']))+'.jpg',frame)
if interrupt & 0xFF == ord('l'):
    cv2.imwrite(os.path.join(directory+'L/'+str(count['l']))+'.jpg',frame)
if interrupt & 0xFF == ord('m'):
    cv2.imwrite(os.path.join(directory+'M/'+str(count['m']))+'.jpg',frame)
if interrupt & 0xFF == ord('n'):
    cv2.imwrite(os.path.join(directory+'N/'+str(count['n']))+'.jpg',frame)
if interrupt & 0xFF == ord('o'):
    cv2.imwrite(os.path.join(directory+'O/'+str(count['o']))+'.jpg',frame)
if interrupt & 0xFF == ord('p'):
    cv2.imwrite(os.path.join(directory+'P/'+str(count['p']))+'.jpg',frame)
```

Model Creation

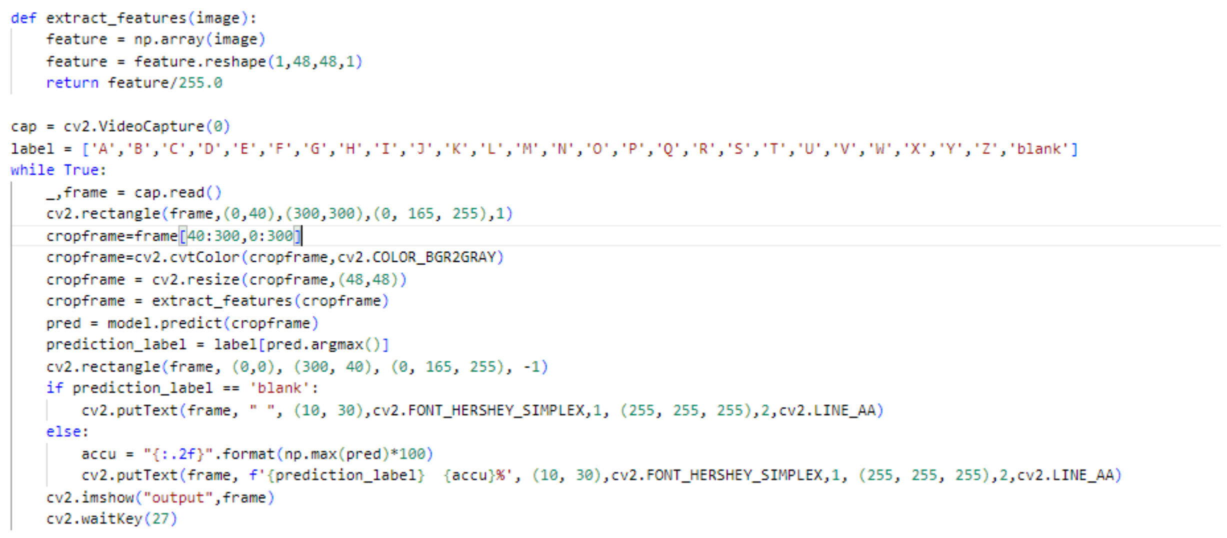


```
train_datagen=ImageDataGenerator(
    rescale=1./255,
)
val_datagen=ImageDataGenerator(rescale=1./255)
batch_size=128

train_generator=train_datagen.flow_from_directory(
    '/content/drive/MyDrive/SignLanguage/Splitdataset48x48/train',
    target_size=(48,48),
    batch_size=batch_size,
    class_mode='categorical',
    color_mode='grayscale'

)
validation_generator=val_datagen.flow_from_directory(
    '/content/drive/MyDrive/SignLanguage/Splitdataset48x48/val',
    target_size=(48,48),
    batch_size=128,
    class_mode='categorical',
    color_mode='grayscale'

)
```

CNN-model

```
model=Sequential()
#convulational layer
model.add(Conv2D(64,kernel_size=(3,3),activation='relu',input_shape=(48,48,1)))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.2))

model.add(Conv2D(128,kernel_size=(3,3),activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.2))

model.add(Conv2D(512,kernel_size=(3,3),activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Flatten())
#connected layer
model.add(Dense(512,activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(256,activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(64,activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(27,activation='softmax'))
```

Realtime Detection



```
def extract_features(image):
    feature = np.array(image)
    feature = feature.reshape(1,48,48,1)
    return feature/255.0

cap = cv2.VideoCapture(0)
label = ['A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z','blank']
while True:
    _,frame = cap.read()
    cv2.rectangle(frame,(0,40),(300,300),(0, 165, 255),1)
    cropframe=frame[40:300,0:300]
    cropframe=cv2.cvtColor(cropframe,cv2.COLOR_BGR2GRAY)
    cropframe = cv2.resize(cropframe,(48,48))
    cropframe = extract_features(cropframe)
    pred = model.predict(cropframe)
    prediction_label = label[pred.argmax()]
    cv2.rectangle(frame, (0,0), (300, 40), (0, 165, 255), -1)
    if prediction_label == 'blank':
        cv2.putText(frame, " ", (10, 30),cv2.FONT_HERSHEY_SIMPLEX,1, (255, 255, 255),2,cv2.LINE_AA)
    else:
        accu = "{:.2f}".format(np.max(pred)*100)
        cv2.putText(frame, f'{prediction_label}  {accu}%', (10, 30),cv2.FONT_HERSHEY_SIMPLEX,1, (255, 255, 255),2,cv2.LINE_AA)
    cv2.imshow("output",frame)
    cv2.waitKey(27)
```

RNN(LSTM)

Necessary Functions

```
def mediapipe_detection(image,model):
    image=cv2.cvtColor(image,cv2.COLOR_BGR2RGB)
    image.flags.writeable=False
    results=model.process(image)
    image.flags.writeable=True
    image=cv2.cvtColor(image,cv2.COLOR_RGB2BGR)
    return image,results
```

```
def draw_styled_landmarks(image,results):
    mp_drawing.draw_landmarks(image,results.face_landmarks,mp_holistic.FACEMESH_CONTOURS,
                            mp_drawing.DrawingSpec(color=(80,110,10),thickness=1 ,circle_radius=1),
                            mp_drawing.DrawingSpec(color=(80,256,121),thickness=1,circle_radius=1))
    mp_drawing.draw_landmarks(image,results.pose_landmarks,mp_holistic.POSE_CONNECTIONS,
                            mp_drawing.DrawingSpec(color=(80,22,10),thickness=1, circle_radius=1),
                            mp_drawing.DrawingSpec(color=(80,44,121),thickness=1,circle_radius=1))
    mp_drawing.draw_landmarks(image,results.left_hand_landmarks,mp_holistic.HAND_CONNECTIONS,
                            mp_drawing.DrawingSpec(color=(121,22,76),thickness=1, circle_radius=1),
                            mp_drawing.DrawingSpec(color=(121,44,250),thickness=1,circle_radius=1))
    mp_drawing.draw_landmarks(image,results.right_hand_landmarks,mp_holistic.HAND_CONNECTIONS,
                            mp_drawing.DrawingSpec(color=(245,117,66),thickness=1 ,circle_radius=1),
                            mp_drawing.DrawingSpec(color=(245,66,230),thickness=1,circle_radius=1))
```

```
cap=cv2.VideoCapture(0)
with mp_holistic.Holistic(min_detection_confidence=0.5,min_tracking_confidence=0.5)as holistic:
    while cap.isOpened():
        ret,frame=cap.read()
        image,results=mediapipe_detection(frame,holistic)
        print(results)
        draw_styled_landmarks(image,results)
        cv2.imshow('OpenCV Feed',image)
        if cv2.waitKey(10)&0xFF==ord('q'):
            break
    cap.release()
    cv2.destroyAllWindows
```

```
import os
import numpy as np

DATA_PATH = r'C:\Users\DELL\HOP_Data'
actions = np.array(['Hello','Thanks','Love You','Yes','No','Help','Please','Food',])
no_sequences = 30
sequence_length=30
start_folder=30
for action in actions:
    for sequence in range(no_sequences):
        try:
            os.makedirs(os.path.join(DATA_PATH, action, str(sequence)))
            print(f"Created directory: {os.path.join(DATA_PATH, action, str(sequence))}")
        except Exception as e:
            print(f"Failed to create directory: {os.path.join(DATA_PATH, action, str(sequence))}")
            print(f"Error: {e}")
```

```
sequences,labels=[],[]
for action in actions:
    for sequence in range(no_sequences):
        window=[]
        for frame_num in range(sequence_length):
            res=np.load(os.path.join(DATA_PATH,action,str(sequence),"{}.npy".format(frame_num)))
            window.append(res)
        sequences.append(window)
        labels.append(label_map[action])
```

Model Creation

```
model=Sequential()
model.add(LSTM(64,return_sequences=True,activation='relu',input_shape=(30,1662)))
model.add(LSTM(128,return_sequences=True,activation='relu'))
model.add(LSTM(64,return_sequences=False,activation='relu'))
model.add(Dense(64,activation='relu'))
model.add(Dense(32,activation='relu'))
model.add(Dense(actions.shape[0],activation='softmax'))
```

```
model.compile(optimizer='Adam',loss='categorical_crossentropy',metrics=['categorical_accuracy'])
```

```
model.fit(X_train,y_train,epochs=1000,validation_data=(X_test,y_test))
```

Realtime detection

```
colors=[(245,117,16),(117,245,16),(16,117,245),(16,117,245),(117,245,16),(117,245,16),(117,245,16),(117,245,16)]
def prob_viz(res,actions,input_frame,colors):
    output_frame=input_frame.copy()
    for num,prob in enumerate(res):
        cv2.rectangle(output_frame,(0,60+num*40),(int(prob*100),90+num*40),colors[num],-1)
        cv2.putText(output_frame,actions[num],(0,85+num*40),cv2.FONT_HERSHEY_SIMPLEX,1,(255,255,255),2,cv2.LINE_AA)
    return output_frame
```

```
sequence=[]
sentence=[]
predictions=[]
threshold=0.45
model.load_weights('Signmain.h5')
cap=cv2.VideoCapture(0)
with mp_holistic.Holistic(min_detection_confidence=0.5,min_tracking_confidence=0.5)as holistic:
    while cap.isOpened():
        ret,frame=cap.read()
        image,results=mediapipe_detection(frame,holistic)
        print(results)
        draw_styled_landmarks(image,results)
        #prediction
        keypoints=extract_keypoints(results)
        sequence.append(keypoints)
        sequence=sequence[-30:]
        if len(sequence)==30:
            res=model.predict(np.expand_dims(sequence,axis=0))[0]
            print(actions[np.argmax(res)])
            predictions.append(np.argmax(res))


        #RESULT PRINTING ON SCREEN
            image=prob_viz(res,actions,image,colors)
            cv2.rectangle(image,(0,0),(640,40),(245,117,16),-1)
            cv2.putText(image,''.join(sentence),(3,30),
                        cv2.FONT_HERSHEY_SIMPLEX,1,(255,255,255),2,cv2.LINE_AA)


        cv2.imshow('OpenCV Feed',image)
        if cv2.waitKey(10)&0xFF==ord('q'):
            break
    cap.release()
    cv2.destroyAllWindows
```