SIGN LANGUAGE TO TEXT CONVERSION

A PROJECT REPORT

Submitted in the partial fulfilment for the award of the degree of

BACHELOR OF ENGINEERING IN ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

Submitted By:

Saransh Gupta (20BCS6662)

Aryan Singh (20BCS6677)

Kailash Dewangan (20BCS6676)

Srikumar Das (20BCS6629)

Under the Supervision of:

Ms. Priyanka Kaushik

CHANDIGARH UNIVERSITY, GHARUAN, MOHALI-140413, PUNJAB

November, 2023

CHANDIGARH UNIVERSITY
Discover. Learn. Empower.

BONAFIDE CERTIFICATE

Certified that this project report **"SIGN LANGUAGE TO TEXT CONVERSION"** is the bonafide work of **"SARANSH GUPTA (20BCS6662), ARYAN SINGH (20BCS6677), KAILASH DEWANGAN (20BCS6676), SRIKUMAR DAS (20BCS6629)"**

who carried out the project work under my/our supervision.

SIGNATURE **SIGNATURE**

HEAD OF THE DEPARTMENT **SUPERVISOR**

INTERNAL EXAMINER **EXTERNAL EXAMINER**

CHANDIGARH UNIVERSITY
Discover. Learn. Empower.

TABLE OF CONTENTS

ABSTRACT

This essay explores the issue of as one of the earliest and most ingrained forms of language for communication, we have created a real-time technique using neural networks. for the finger-spelled American sign language. We propose to identify human hand gestures from a camera-captured image using a convolution neural network. The objective is to identify hand gestures utilized in human task activities from a camera image. The hand position and orientation are employed to collect the training and testing data for the CNN. After the hand has been placed through a filter, it is next put through a classifier, which establishes what class the hand movements fall under.

Sign language is one of the oldest and most natural form of language for communication, but since most people do not know sign language and interpreters are very difficult to come by, we have come up with a real time method using neural networks for fingerspelling based American sign language. In our method, the hand is first passed through a filter and after the filter is applied the hand is passed through a classifier which predicts the class of the hand gestures. Our method provides 95.7 % accuracy for the 26 letters of the alphabet.

Keywords: sign language, automated, ML Models, monitoring.

GRAPHICAL ABSTRACT

Sign language is a visual language that uses hand gestures, facial expressions, and body language to convey meaning. It is the primary means of communication for millions of people worldwide who are deaf or hard of hearing. Sign language to text conversion (SLTC) is a technology that can automatically translate sign language gestures into text.

Challenges

SLTC is a challenging task due to the complex and nuanced nature of sign language. Sign language is not simply a visual representation of spoken language; it has its own grammar, syntax, and vocabulary. Additionally, sign language interpretation can vary depending on the individual signer and the context of the conversation.

Approaches

There are two main approaches to SLTC: data-driven and knowledge-driven. Data-driven approaches use large datasets of signed and spoken language to train machine learning algorithms to recognize sign language gestures. Knowledge-driven approaches use handcrafted rules and models to represent the grammar and syntax of sign language.

Applications

SLTC has a wide range of potential applications, including:

Providing real-time communication access for deaf and hard of hearing people

Making educational materials and resources accessible to deaf and hard of hearing students

Enabling deaf and hard of hearing people to participate in online communities and forums

Improving the accessibility of public spaces and services for deaf and hard of hearing people

Future Directions

SLTC is a rapidly developing field of research. As machine learning algorithms become more sophisticated and as datasets of signed and spoken language grow, we can expect to see SLTC technology become more accurate and widely used.

Conclusion

SLTC is a promising technology that has the potential to revolutionize the lives of deaf and hard of hearing people. As research in this area continues to advance, we can expect to see even more innovative and effective SLTC solutions emerge in the future.

Key Figures

The number of people who are deaf or hard of hearing worldwide is estimated to be 466 million.

The global market for SLTC is expected to reach $1.8 billion by 2027.

Chapter 1: INTRODUCTION

American sign language is a predominant sign language Since the only disability Deaf and Dumb (hereby referred to as D&M) people have is communication related and since they cannot use spoken languages, the only way for them to communicate is through sign language. Communication is the process of exchange of thoughts and messages in various ways such as speech, signals, behavior and visuals. D&M people make use of their hands to express different gestures to express their ideas with other people. Gestures are the non-verbally exchanged messages and these gestures are understood with vision. This nonverbal communication of deaf and dumb people is called sign language. A sign language is a language which uses gestures instead of sound to convey meaning combining hand-shapes, orientation and movement of the hands, arms or body, facial expressions and lip-patterns. Contrary to popular belief, sign language is not international. These vary from region to region.

Sign language is a visual language and consists of 3 major components [6]:

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.

Figure -1

Minimizing the verbal exchange gap among D&M and non-D&M people turns into a want to make certain effective conversation among all. Sign language translation is among one of the most growing lines of research and it enables the maximum natural manner of communication for those with hearing impairments. A hand gesture recognition system offers an opportunity for deaf people to talk with vocal humans without the need of an interpreter. The system is built for the automated conversion of ASL into textual content and speech.

In our project we primarily focus on producing a model which can recognize Fingerspelling based hand gestures in order to form a complete word by combining each gesture. The gestures we aim to train are as given in the image below.

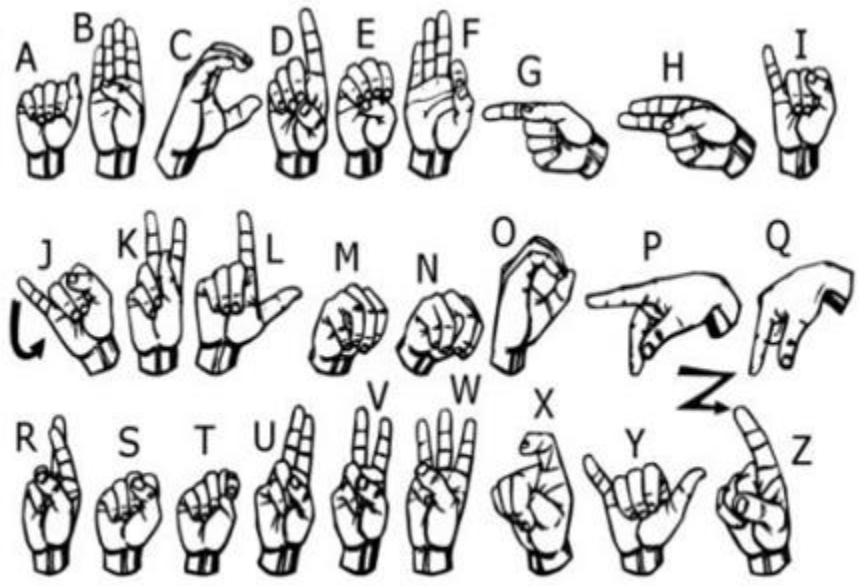

Figure - 2

We have developed a real-time method employing neural networks because sign language is one of the oldest and most innate forms of language for communication. for American sign language that uses finger spelling. We suggest using a convolution neural network to identify human hand motions from a camera-captured image.

Recognizing hand movements used in human task activities from a camera image is the goal. To gather the training and testing data for the CNN, the hand position and orientation are used.

The hand is first put through a filter, and once that has been done, it is put through a classifier, which determines what class the hand movements belong to.

1.1 PROBLEM DEFINITION

The central idea of this project is to work with the data that namely consists of training data (Data of financial transactions) that are inherent to the system in which you want to do credit card fraud detection. After the process of learning through training data, the models are made using different machine learning techniques and should be able to classify the fraudulent and legit transactions

The advent of credit cards and their increasing functionality have not only given people more personal comfort but have also attracted malicious characters interested in the handsome rewards to be earned. In a very short time, a huge amount of money can be earned without taking too many risks and this results in delayed discovery of the fraud. When banks lose money due to credit card fraud1 card holders partially (possibly entirely) pay for the loss through higher interest rates1 higher member-ship fees1 and reduced benefits. Hence1 it is both the banks' and card holders' interest to reduce illegitimate use of credit cards and that is the reason why financial institutions started to do fraud detection.

- Approximately two million individuals worldwide are deaf; they are excluded from many social activities and underappreciated as a communication issue in society.
- SLR can be a useful interpreter that benefits the deaf population regardless of their gender.

1.2 PROJECT OVERVIEW

- Approximately two million individuals worldwide are deaf; they are excluded from many social activities and underappreciated as a communication issue in society.
- SLR can be a useful interpreter that benefits the deaf population regardless of their gender.

1.3 HARDWARE SPECIFICATIONS

1.3.1 PC

A pc is a personal computer that can be used for multiple purposes depending on its size, capabilities, and price. They are to be operated directly by the end-user. Personal computers are single-user systems and are portable. Our web application program will be installed on the pc for our clients to use it. This makes it feasible for individual use.

1.4 SOFTWARE SPECIFICATIONS

1.4.1Jupyter Notebook:

Jupyter Notebook is a web-based open-source application that is used for editing, creating running, and sharing documents that contain live codes, visualization, text, and equations. Its core supported programming languages are Julia, R, and Python. Jupyter notebook comes with an IPython kernel that allows the programmer to write programs in python. There are over 100 kernels other than IPython available for use.

1.4.2 Tenserflow

Google created the open-source machine learning framework known as TensorFlow. It is frequently employed for creating and refining deep learning and machine learning models. The Python integration, adaptability, scalability, and broad community support of TensorFlow are well-known features. For creating and deploying machine learning applications, it offers both high-level and low-level APIs.

1.4.3 Keras

An interface for creating and training neural networks is provided by the open-source deep learning framework Keras. Since TensorFlow 2.0, Keras has been incorporated into TensorFlow after initially being created as a separate library. It is renowned for its user-friendly, high-level API, which makes it simple for developers to construct and test neural networks in a clear and

succinct manner. Many complicated deep learning ideas are abstracted in Keras, which also offers a straightforward yet effective tool for creating artificial neural networks. This makes Keras a favorite among both newcomers and seasoned machine learning professionals.

1.4.4 OpenCV

An open-source library for computer vision and image processing is called OpenCV, or Open Source Computer Vision Library. It offers a large array of functions, tools, and algorithms for work in computer vision, image analysis, and machine learning. OpenCV is very adaptable and is utilized in many different industries, such as robots, augmented reality, and medical image processing. Numerous programming languages, including Python and C++, are supported, making it usable by a variety of developers. OpenCV is a popular choice for many computer vision projects and applications because of its effectiveness, portability, and thorough documentation.

Chapter 2: LITERATURE REVIEW

2.1 Existing System Summary:

In the recent years there has been tremendous research done on the hand gesture recognition. With the help of literature survey, we realized that the basic steps in hand gesture recognition are:

- Data acquisition
- Data pre-processing
- Feature extraction
- Gesture classification

2.2 Data acquisition:

The different approaches to acquire data about the hand gesture can be done in the following ways:

1. **Use of sensory devices:**

   It uses electromechanical devices to provide exact hand configuration, and position. Different glove-based approaches can be used to extract information. But it is expensive and not user friendly.

2. **Vision based approach:**

   In vision-based methods, the computer webcam is the input device for observing the information of hands and/or fingers. The Vision Based methods require only a camera, thus realizing a natural interaction between humans and computers without the use of any extra devices, thereby reducing cost. These systems tend to complement biological vision by describing artificial vision systems that are implemented in software and/or hardware. The main challenge of vision-based hand detection ranges from coping with the large variability of the human hand's appearance due to a huge number of hand movements, to different skin-color possibilities as well as to the variations in viewpoints, scales, and speed of the camera capturing the scene.

2.3 Data Pre-Processing and 2.4 Feature extraction for vision-based approach:

- In [1] the approach for hand detection combines threshold-based colour detection with background subtraction. We can use AdaBoost face detector to differentiate between faces and hands as they both involve similar skin-color.

- We can also extract necessary image which is to be trained by applying a filter called Gaussian Blur (also known as Gaussian smoothing). The filter can be easily applied using open computer vision (also known as OpenCV) and is described in [3].

- For extracting necessary image which is to be trained we can use instrumented gloves as mentioned in [4]. This helps reduce computation time for Pre-Processing and gives us more concise and accurate data compared to applying filters on data received from video extraction.

- We tried doing the hand segmentation of an image using color segmentation techniques but skin colorur and tone is highly dependent on the lighting conditions due to which output, we got for the segmentation we tried to do were no so great. Moreover, we have a huge number of symbols to be trained for our project many of which look similar to each other like the gesture for symbol ‘V’ and digit ‘2’, hence we decided that in order to produce better accuracies for our large number of symbols, rather than segmenting the hand out of a random background we keep background of hand a stable single colour so that we don’t need to segment it on the basis of skin colour. This would help us to get better results.

2.5 Gesture Classification:

- In [1] Hidden Markov Models (HMM) is used for the classification of the gestures. This model deals with dynamic aspects of gestures. Gestures are extracted from a sequence of video images by tracking the skin-color blobs corresponding to the hand into a body– face space centred on the face of the user.

- The goal is to recognize two classes of gestures: deictic and symbolic. The image is filtered using a fast look–up indexing table. After filtering, skin colour pixels are gathered into blobs. Blobs are statistical objects based on the location (x, y) and the colorimetry (Y, U, V) of the skin color pixels in order to determine homogeneous areas.

- In [2] Naïve Bayes Classifier is used which is an effective and fast method for static hand gesture recognition. It is based on classifying the different gestures according to geometric based invariants which are obtained from image data after segmentation.

- Thus, unlike many other recognition methods, this method is not dependent on skin colour. The gestures are extracted from each frame of the video, with a static background. The first step is to segment and label the objects of interest and to extract geometric invariants from them. Next step is the classification of gestures by using a K nearest neighbor algorithm aided with distance weighting algorithm (KNNDW) to provide suitable data for a locally weighted Naïve Bayes" classifier.

- According to the paper on "Human Hand Gesture Recognition Using a Convolution Neural Network" by Hsien-I Lin, Ming-Hsiang Hsu, and Wei-Kai Chen (graduates of Institute of Automation Technology National Taipei University of Technology Taipei, Taiwan), they have constructed a skin model to extract the hands out of an image and then apply binary threshold to the whole image. After obtaining the threshold image they calibrate it about the principal axis in order to centre the image about the axis. They input this image to a convolutional neural network model in order to train and predict the outputs. They have trained their model over 7 hand gestures and using this model they produced an accuracy of around 95% for those 7 gestures.

2.6 Proposed System:

➢ This project mainly focuses on developing a system that can do sign language to text conversion.

➢ The Sign Language to Text Conversion System aims to bridge communication gaps by providing a seamless transition from sign language gestures to written text. Leveraging advanced technologies, this system empowers individuals with hearing impairments to communicate effectively in various settings.

➢ Utilizes a high-quality camera or sensor array to capture sign language gestures. Implements computer vision algorithms for real-time recognition of signs.

➢ Converts recognized signs into written text, considering grammar and contextual cues. Incorporates natural language processing (NLP) techniques for improved linguistic accuracy.

Chapter 3: KEY WORDS AND DEFINITION

3.1 Feature Extraction and Representation:

The representation of an image as a 3D matrix having dimension as of height and width of the image and the value of each pixel as depth (1 in case of Grayscale and 3 in case of RGB). Further, these pixel values are used for extracting useful features using CNN.

3.2 Artificial Neural Network (ANN):

Artificial Neural Network 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.

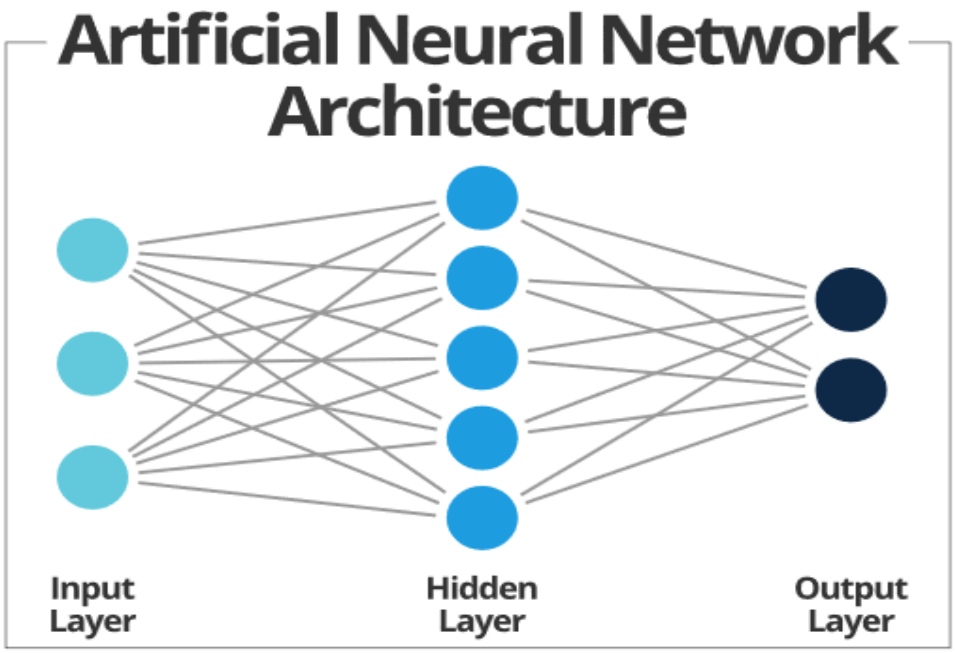

Figure - 3

These are capable of learning and have to be trained. There are different learning strategies:

- Unsupervised Learning
- Supervised Learning
- Reinforcement Learning

3.3 Convolutional Neural Network (CNN):

Unlike regular Neural Networks, in the layers of CNN, the neurons are arranged in 3 dimensions: width, height, depth. The neurons in a layer will only be connected to a small region of the layer (window size) before it, instead of all of the neurons in a fully-connected manner. Moreover, the final output layer would have dimensions (number of classes), because by the end of the CNN architecture we will reduce the full image into a single vector of class scores.

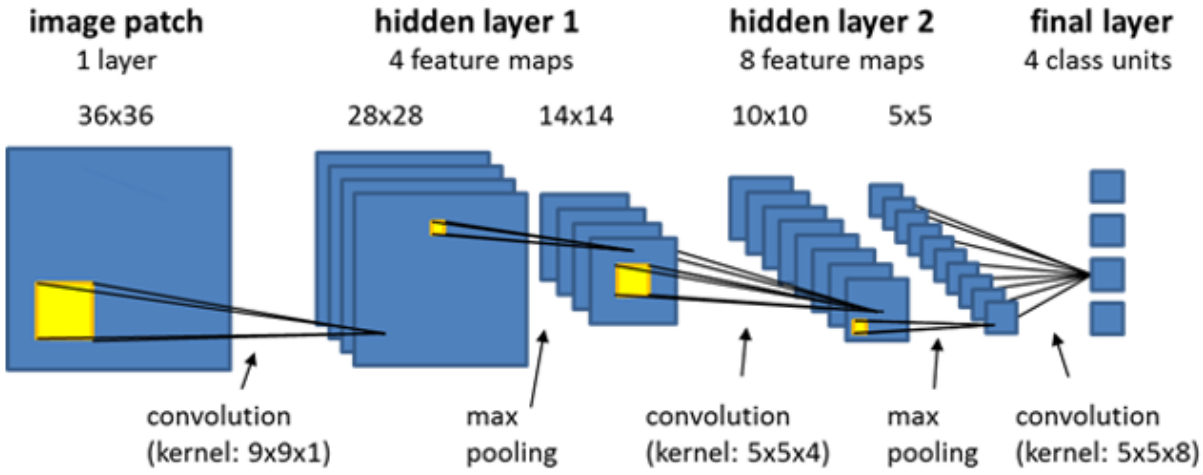

Figure – 4

1. Convolution Layer:

In convolution layer we take a small window size [typically of length 5*5] that extends to the depth of the input matrix. The layer consists of learnable filters of window size. During every iteration we slid the window by stride size [typically 1], and compute the dot product of filter entries and input values at a given position.

As we continue this process we will create a 2-Dimensional activation matrix that gives the response of that matrix at every spatial position. That is, the network will learn filters that activate when they see some type of visual feature such as an edge of some orientation or a blotch of some colour.

2. **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 advantage of of all Values in a window.

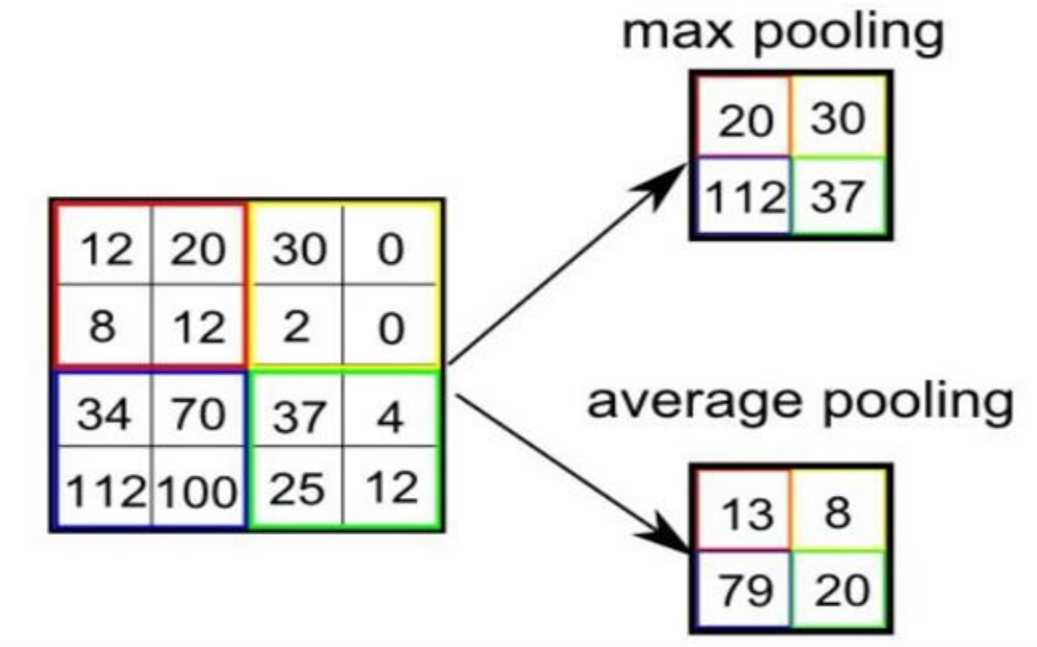

Figure - 5

3. Fully Connected Layer:

In convolution layer, neurons are connected only to a local region, while in a fully connected region, we will connect all the inputs to neurons.

4. Final Output Layer:

After getting values from fully connected layer, we will connect them to the 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.

3.4 TensorFlow:

TensorFlow is an end-to-end open-source platform for Machine Learning. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in Machine Learning and developers easily build and deploy Machine Learning powered applications.

TensorFlow offers multiple levels of abstraction so you can choose the right one for your needs. Build and train models by using the high-level Keras API, which makes getting started with TensorFlow and machine learning easy.

If you need more flexibility, eager execution allows for immediate iteration and intuitive debugging. For large ML training tasks, use the Distribution Strategy API for distributed training on different hardware configurations without changing the model definition.

3.5 Keras:

Keras is a high-level neural networks library written in python that works as a wrapper to TensorFlow. It is used in cases where we want to quickly build and test the neural network with minimal lines of code. It contains implementations of commonly used neural network elements like layers, objective, activation functions, optimizers, and tools to make working with images and text data easier.

3.6 OpenCV:

OpenCV (Open-Source Computer Vision) is an open-source library of programming functions used for real-time computer-vision.

It is mainly used for image processing, video capture and analysis for features like face and object recognition. It is written in C++ which is its primary interface, however bindings are available for Python, Java, MATLAB/OCTAVE.

Chapter 4: OBJECTIVES

Accurate Gesture Recognition:

Develop and implement a robust computer vision system for precise and real-time recognition of sign language gestures.

Machine Learning Integration:

Employ machine learning algorithms to continuously improve the system's ability to adapt and recognize variations in signing styles, ensuring high accuracy.

Multi-Lingual Support:

Design the system to accommodate various sign languages, addressing the linguistic diversity within the community of individuals with hearing impairments.

Real-Time Translation:

Enable the system to provide instantaneous translation of sign language gestures into written text, allowing for seamless communication.

User-Friendly Interface:

Create an intuitive and accessible user interface that accommodates individuals with varying levels of technological proficiency and provides a positive user experience.

Customization Options:

Implement features that allow users to customize the system according to their preferences, such as adjusting font size, color, and style for the displayed text.

Chapter 5: METHODOLOGY

The system is a vision-based approach. All signs are represented with bare hands and so it eliminates the problem of using any artificial devices for interaction.

5.1 Data Set Generation:

For the project we tried to find already made datasets but we couldn't find dataset in the form of raw images that matched our requirements. All we could find were the datasets in the form of RGB values. Hence, we decided to create our own data set. Steps we followed to create our data set are as follows.

We used Open computer vision (OpenCV) library in order to produce our dataset.

Firstly, we captured around 800 images of each of the symbol in ASL (American Sign Language) for training purposes and around 200 images per symbol for testing purpose.

First, we capture each frame shown by the webcam of our machine. In each frame we define a Region Of Interest (ROI) which is denoted by a blue bounded square as shown in the image below:

Then, we apply Gaussian Blur Filter to our image which helps us extract various features of our image. The image, after applying Gaussian Blur, looks as follows:

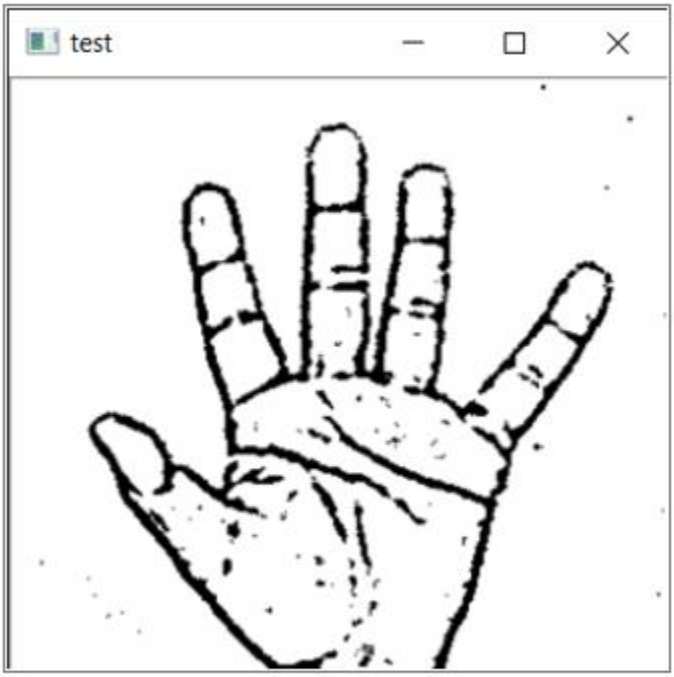

Figure – 8

5.2 Gesture Classification:

Our approach uses two layers of algorithm to predict the final symbol of the user.

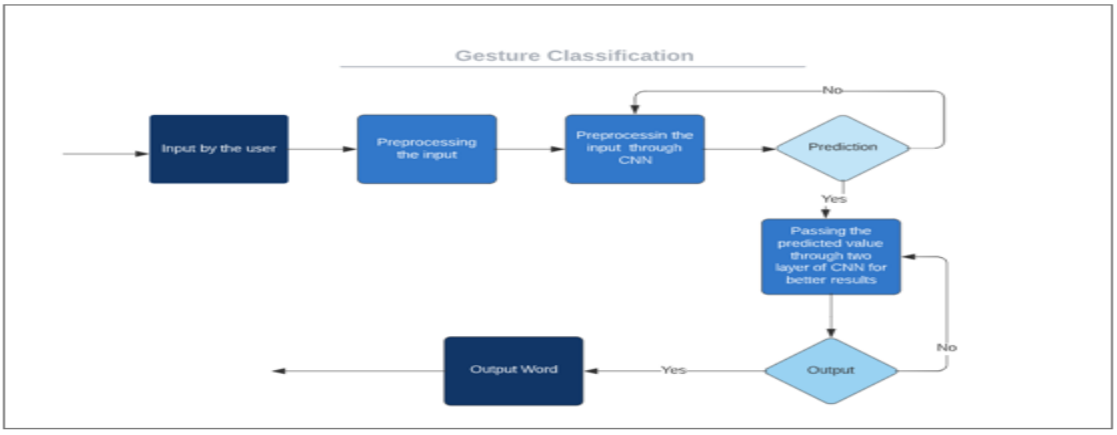

Figure - 9

Algorithm Layer 1:

1. Apply Gaussian Blur filter and threshold to the frame taken with openCV to get the processed image after feature extraction.
2. This processed image is passed to the CNN model for prediction and if a letter is detected for more than 50 frames then the letter is printed and taken into consideration for forming the word.
3. Space between the words is considered using the blank symbol.

Algorithm Layer 2:

1. We detect various sets of symbols which show similar results on getting detected.
2. We then classify between those sets using classifiers made for those sets only.

Layer 1:

- **CNN Model:**

1. **1st Convolution Layer:** The input picture has resolution of 128x128 pixels. It is first processed in the first convolutional layer using 32 filter weights (3x3 pixels each). This will result in a 126X126 pixel image, one for each Filter-weights.
2. **1st Pooling Layer:** The pictures are down sampled using max pooling of 2x2 i.e we keep the highest value in the 2x2 square of array. Therefore, our picture is down sampled to 63x63 pixels.
3. **2nd Convolution Layer:** Now, these 63 x 63 from the output of the first pooling layer is served as an input to the second convolutional layer. It is processed in the second convolutional layer using 32 filter weights (3x3 pixels each). This will result in a 60 x 60 pixel image.
4. **2nd Pooling Layer:** The resulting images are down sampled again using max pool of 2x2 and is reduced to 30 x 30 resolution of images.
5. **1st Densely Connected Layer:** Now these images are used as an input to a fully connected layer with 128 neurons and the output from the second convolutional layer is reshaped to an

array of 30x30x32 =28800 values. The input to this layer is an array of 28800 values. The output of these layer is fed to the 2nd Densely Connected Layer. We are using a dropout layer of value 0.5 to avoid overfitting.

6. **2nd Densely Connected Layer:** Now the output from the 1st Densely Connected Layer is used as an input to a fully connected layer with 96 neurons.
7. **Final layer:** The output of the 2nd Densely Connected Layer serves as an input for the final layer which will have the number of neurons as the number of classes we are classifying (alphabets + blank symbol).

- **Activation Function:**

  We have used ReLU (Rectified Linear Unit) in each of the layers (convolutional as well as fully connected neurons).

  ReLU calculates max(x,0) for each input pixel. This adds nonlinearity to the formula and helps to learn more complicated features. It helps in removing the vanishing gradient problemand speeding up the training by reducing the computation time.

- **Pooling Layer:**

  We apply **Max** pooling to the input image with a pool size of (2, 2) with ReLU activation function. This reduces the amount of parameters thus lessening the computation cost and reduces overfitting.

- **Dropout Layers:**

  The problem of overfitting, where after training, the weights of the network are so tuned to the training examples they are given that the network doesn't perform well when given new examples. This layer "drops out" a random set of activations in that layer by setting them to zero. The network should be able to provide the right classification or output for a specific example even if some of the activations are dropped out [5].

- **Optimizer:**

  We have used Adam optimizer for updating the model in response to the output of the loss function.

  Adam optimizer combines the advantages of two extensions of two stochastic gradient descent algorithms namely adaptive gradient algorithm (ADA GRAD) and root mean square propagation (RMSProp).

Layer 2:

We are using two layers of algorithms to verify and predict symbols which are more similar to each other so that we can get us close as we can get to detect the symbol shown. In our testing we found that following symbols were not showing properly and were giving other symbols also:

1. **For D : R and U**
2. **For U : D and R**
3. **For I : T, D, K and I**
4. **For S : M and N**

So, to handle above cases we made three different classifiers for classifying these sets:

1. **{D, R, U}**
2. **{T, K, D, I}**
3. **{S, M, N}**

5.3 Finger Spelling Sentence Formation Implementation:

1. Whenever the count of a letter detected exceeds a specific value and no other letter is close to it by a threshold, we print the letter and add it to the current string (In our code we kept the value as 50 and difference threshold as 20).
2. Otherwise, we clear the current dictionary which has the count of detections of present symbol to avoid the probability of a wrong letter getting predicted.
3. Whenever the count of a blank (plain background) detected exceeds a specific value and if the current buffer is empty no spaces are detected.

4. In other case it predicts the end of word by printing a space and the current gets appended to the sentence below.

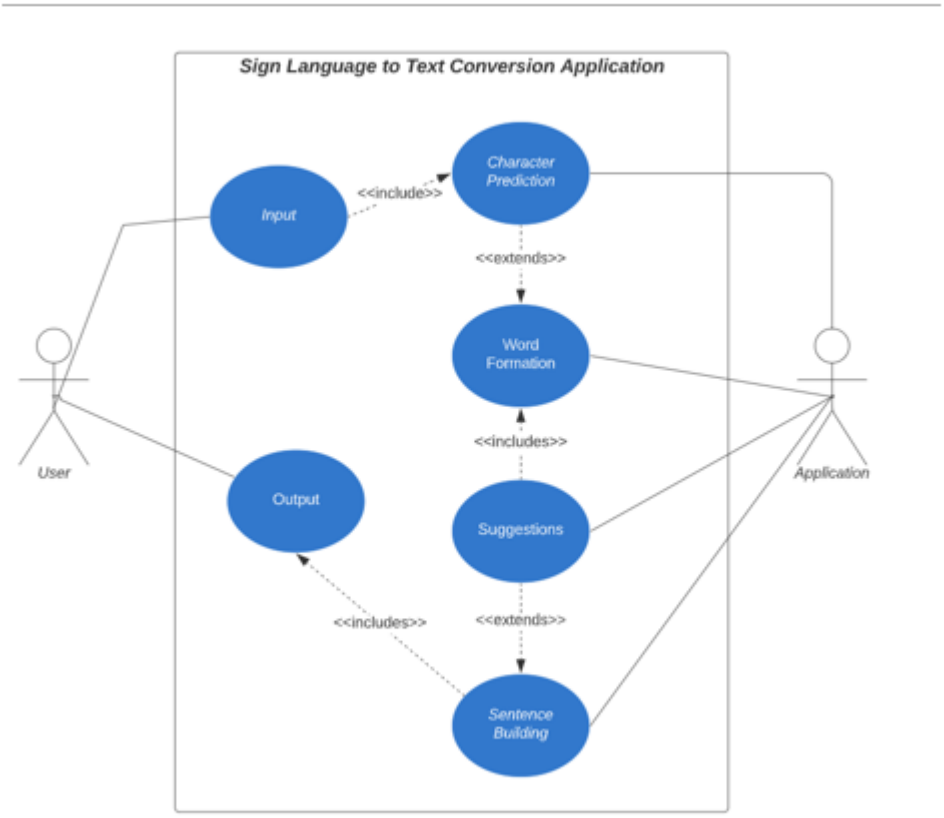

Figure - 10

5.4 AutoCorrect Feature:

A python library **Hunspell_suggest** is used to suggest correct alternatives for each (incorrect) input word and we display a set of words matching the current word in which the user can select a word to append it to the current sentence. This helps in reducing mistakes committed in spellings and assists in predicting complex words.

5.5 Training and Testing:

We convert our input images (RGB) into grayscale and apply gaussian blur to remove unnecessary noise. We apply adaptive threshold to extract our hand from the background and resize our images to 128 x 128.

We feed the input images after pre-processing to our model for training and testing after applying all the operations mentioned above.

The prediction layer estimates how likely the image will fall under one of the classes. So, the output is normalized between 0 and 1 and such that the sum of each value in each class sums to 1. We have achieved this using SoftMax function.

At first the output of the prediction layer will be somewhat far from the actual value. To make it better we have trained the networks using labelled data. The cross-entropy is a performance measurement used in the classification. It is a continuous function which is positive at values which is not same as labelled value and is zero exactly when it is equal to the labelled value. Therefore, we optimized the cross-entropy by minimizing it as close to zero. To do this in our network layer we adjust the weights of our neural networks. TensorFlow has an inbuilt function to calculate the cross entropy.

As we have found out the cross-entropy function, we have optimized it using Gradient Descent in fact with the best gradient descent optimizer is called Adam Optimizer.

6. Challenges Faced:

There were many challenges faced during the project. The very first issue we faced was that concerning the data set. We wanted to deal with raw images and that too square images as CNN in Keras since it is much more convenient working with only square images.

We couldn't find any existing data set as per our requirements and hence we decided to make our own data set. Second issue was to select a filter which we could apply on our images so that proper features of the images could be obtained and hence then we could provide that image as input for CNN model.

We tried various filters including binary threshold, canny edge detection, Gaussian blur etc. but finally settled with Gaussian Blur Filter.

More issues were faced relating to the accuracy of the model we had trained in the earlier phases. This problem was eventually improved by increasing the input image size and also by improving the data set.

7. Results:

We have achieved an accuracy of **95.8%** in our model using only layer 1 of our algorithm, and using the combination of **layer 1 and layer 2** we achieve an accuracy of **98.0%**, which is a better accuracy then most of the current research papers on American sign language.

Most of the research papers focus on using devices like Kinect for hand detection.

In [7] they build a recognition system for Flemish sign language using convolutional neural networks and Kinect and achieve an error rate of **2.5%.**

In [8] a recognition model is built using hidden Markov model classifier and a vocabulary of 30 words and they achieve an error rate of **10.90%**.

In [9] they achieve an average accuracy of **86%** for 41 static gestures in Japanese sign language.

Using depth sensors map [10] achieved an accuracy of **99.99%** for observed signers and **83.58%** and **85.49%** for new signers.

They also used CNN for their recognition system. One thing should be noted that our model doesn't uses any background subtraction algorithm whiles some of the models present above do that.

So, once we try to implement background subtraction in our project the accuracies may vary. On the other hand, most of the above projects use Kinect devices but our main aim was to create a project which can be used with readily available resources. A sensor like Kinect not only isn't readily available but also is expensive for most of audience to buy and our model uses a normal webcam of the laptop hence it is great plus point.

Below are the confusion matrices for our results.

		Predicted Values																							
		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
	A	147	0	0	0	0	0	0	0	0	0	0	0	1	2	0	0	0	0	0	0	0	0	2	0	0
	B	0	139	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	11	0	0	0
	C	0	0	152	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	D	0	0	0	145	0	0	0	0	0	0	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	E	0	0	0	0	152	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	F	0	0	0	0	0	135	0	0	0	0	0	4	0	0	0	0	0	1	0	0	2	10	0	0	0
C	G	0	0	0	0	0	0	150	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
o	H	1	0	0	0	0	0	7	143	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	1
r	I	0	0	0	33	0	0	0	0	108	0	2	0	0	0	0	0	0	0	0	7	1	0	0	0	0
r	J	0	0	0	0	0	0	0	0	0	153	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
e	K	0	0	0	0	0	0	0	0	0	0	153	0	0	0	0	0	0	0	0	0	0	0	0	0	0
c	L	0	0	0	0	0	0	0	0	0	0	0	153	0	0	0	0	0	0	0	0	0	0	0	0	0
t	M	0	0	0	0	0	0	0	0	0	0	2	0	152	0	0	0	0	0	0	0	0	0	0	0	0
	N	0	0	0	0	0	0	0	0	0	0	0	0	0	152	0	0	0	0	0	0	0	0	0	0	0
V	O	0	0	0	0	0	0	0	0	0	0	0	0	0	0	154	0	0	0	0	0	0	0	0	0	0
a	P	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	153	0	0	0	0	0	0	0	0	0
l	Q	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	2	147	1	0	0	0	0	0	0	0
u	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	150	0	0	0	0	0	0	0
e	S	0	0	0	0	1	0	0	0	0	0	0	0	0	1	10	0	0	0	132	0	0	0	0	8	0
s	T	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	151	0	0	0	0	0
	U	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	35	0	0	115	0	0	0	0
	V	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	151	1	0	0
	W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	149	0	0
	X	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	148	0
	Y	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	151
	Z	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Algo 1

Figure - 11

		Predicted Values																							
		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
	A	147	0	0	0	0	0	0	0	0	0	0	0	1	2	0	0	0	0	0	0	0	0	2	0	0
	B	0	139	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	11	0	0	0
	C	0	0	152	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	D	0	0	0	153	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	E	0	0	0	0	152	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	F	0	0	0	0	0	135	0	0	0	0	0	4	0	0	0	0	0	0	0	0	3	10	0	0	0
C	G	0	0	0	0	0	0	150	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
o	H	1	0	0	0	0	0	7	143	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	1
r	I	0	0	0	0	0	0	0	0	150	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
r	J	0	0	0	0	0	0	0	0	0	153	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
e	K	0	0	0	0	0	0	0	0	0	0	153	0	0	0	0	0	0	0	0	0	0	0	0	0	0
c	L	0	0	0	0	0	0	0	0	0	0	0	153	0	0	0	0	0	0	0	0	0	0	0	0	0
t	M	0	0	0	0	0	0	0	0	0	0	2	0	152	0	0	0	0	0	0	0	0	0	0	0	0
	N	0	0	0	0	0	0	0	0	0	0	0	0	0	152	0	0	0	0	0	0	0	0	0	0	0
V	O	0	0	0	0	0	0	0	0	0	0	0	0	0	0	154	0	0	0	0	0	0	0	0	0	0
a	P	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	153	0	0	0	0	0	0	0	0	0
l	Q	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	2	147	1	0	0	0	0	0	0	0
u	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	150	0	0	0	0	0	0	0
e	S	0	0	0	0	1	0	0	0	0	0	0	0	0	0	10	0	0	0	133	0	0	0	0	8	0
s	T	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	151	0	0	0	0	0
	U	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	150	0	0	0	0
	V	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	151	1	0	0
	W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	149	0	0
	X	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	148	0
	Y	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	151
	Z	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Algo 1 + Algo 2

Figure – 12

8. Conclusion:

In this report, a functional real time vision based American Sign Language recognition for D&M people have been developed for asl alphabets.

We achieved final accuracy of **98.0%** on our data set. We have improved our prediction after implementing two layers of algorithms wherein we have verified and predicted symbols which are more similar to each other.

This gives us the ability to detect almost all the symbols provided that they are shown properly, there is no noise in the background and lighting is adequate.

9. Future Scope:

We are planning to achieve higher accuracy even in case of complex backgrounds by trying out various background subtraction algorithms.

We are also thinking of improving the Pre-Processing to predict gestures in low light conditions with a higher accuracy.

This project can be enhanced by being built as a web/mobile application for the users to conveniently access the project. Also, the existing project only works for ASL; it can be extended to work for other native sign languages with the right amount of data set and training. This project implements a finger spelling translator; however, sign languages are also spoken in a contextual basis where each gesture could represent an object, or verb. So, identifying this kind of a contextual signing would require a higher degree of processing and natural language processing (NLP).

CODE AND OUTPUTS:

```
import cv2
from cvzone.HandTrackingModule import HandDetector
from cvzone.ClassificationModule import Classifier
import numpy as np
import os, os.path
from keras.models import load_model
import traceback

capture = cv2.VideoCapture(0)

hd = HandDetector(maxHands=1)
hd2 = HandDetector(maxHands=1)
# #training data
# count = len(os.listdir("D://sign2text_dataset_2.0/Binary_imgs//A"))

#testing data
#count = len(os.listdir("D://test_data_2.0//Gray_imgs//A"))


p_dir = "A"
c_dir = "a"

offset = 30
step = 1
flag=False
suv=0
#C:\Users\devansh raval\PycharmProjects\pythonProject
white=np.ones((400,400),np.uint8)*255
cv2.imwrite("C:\\Users\\saran\\Desktop\\New folder\\titu mama\\Sign-Language-To-Text-and-Speech-Conversion\\white.jpg",white)


while True:
    try:
        _, frame = capture.read()
        frame = cv2.flip(frame, 1)
        hands= hd.findHands(frame, draw=False, flipType=True)
        img_final=img_final1=img_final2=0
```

```
        if hands:
            hand = hands[0]
            x, y, w, h = hand['bbox']
            image = frame[y - offset:y + h + offset, x - offset:x + w + offset]
            #image1 = imgg[y - offset:y + h + offset, x - offset:x + w + offset]



            roi = image     #rgb image without drawing
           # roi1 = image1   #rdb image with drawing



            # #for simple gray image without draw
            gray = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY)
            blur = cv2.GaussianBlur(gray, (1, 1), 2)
            #

            # #for binary image
            gray2 = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY)
            blur2 = cv2.GaussianBlur(gray2, (5, 5), 2)
            th3 = cv2.adaptiveThreshold(blur2, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY_INV, 11, 2)
            ret, test_image = cv2.threshold(th3, 27, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)
            #
            #
            test_image1=blur
            img_final1 = np.ones((400, 400), np.uint8) * 148
            h = test_image1.shape[0]
            w = test_image1.shape[1]
            img_final1[((400 - h) // 2):((400 - h) // 2) + h, ((400 - w) // 2):((400 - w) // 2) + w] = test_image1

            img_final = np.ones((400, 400), np.uint8) * 255
            h = test_image.shape[0]
            w = test_image.shape[1]
            img_final[((400 - h) // 2):((400 - h) // 2) + h, ((400 - w) // 2):((400 - w) // 2) + w] = test_image
```

```
    hands = hd.findHands(frame, draw=False, flipType=True)

    if hands:
        # #print(" --------- lmlist=",hands[1])
        hand = hands[0]
        x, y, w, h = hand['bbox']
        image = frame[y - offset:y + h + offset, x - offset:x + w + offset]
        white = cv2.imread("C:\\Users\\saran\\Desktop\\New folder\\titu mama\\Sign-Language-To-Text-and-Speech-Conversion\\white.jpg")
        # img_final=img_final1=img_final2=0
        handz = hd2.findHands(image, draw=False, flipType=True)
        if handz:
            hand = handz[0]
            pts = hand['lmList']
            # x1,y1,w1,h1=hand['bbox']

            os = ((400 - w) // 2) - 15
            os1 = ((400 - h) // 2) - 15
            for t in range(0, 4, 1):
                cv2.line(white, (pts[t][0] + os, pts[t][1] + os1), (pts[t + 1][0] + os, pts[t + 1][1] + os1),
                         (0, 255, 0), 3)
            for t in range(5, 8, 1):
                cv2.line(white, (pts[t][0] + os, pts[t][1] + os1), (pts[t + 1][0] + os, pts[t + 1][1] + os1),
                         (0, 255, 0), 3)
            for t in range(9, 12, 1):
                cv2.line(white, (pts[t][0] + os, pts[t][1] + os1), (pts[t + 1][0] + os, pts[t + 1][1] + os1),
                         (0, 255, 0), 3)
            for t in range(13, 16, 1):
                cv2.line(white, (pts[t][0] + os, pts[t][1] + os1), (pts[t + 1][0] + os, pts[t + 1][1] + os1),
                         (0, 255, 0), 3)
            for t in range(17, 20, 1):
                cv2.line(white, (pts[t][0] + os, pts[t][1] + os1), (pts[t + 1][0] + os, pts[t + 1][1] + os1),
                         (0, 255, 0), 3)
            cv2.line(white, (pts[5][0] + os, pts[5][1] + os1), (pts[9][0] + os, pts[9][1] + os1), (0, 255, 0),
                     3)
            cv2.line(white, (pts[9][0] + os, pts[9][1] + os1), (pts[13][0] + os, pts[13][1] + os1), (0, 255, 0),
                     3)
            cv2.line(white, (pts[13][0] + os, pts[13][1] + os1), (pts[17][0] + os, pts[17][1] + os1),
                     (0, 255, 0), 3)
            cv2.line(white, (pts[0][0] + os, pts[0][1] + os1), (pts[5][0] + os, pts[5][1] + os1), (0, 255, 0),
                     3)
```

```
            cv2.line(white, (pts[0][0] + os, pts[0][1] + os1), (pts[17][0] + os, pts[17][1] + os1), (0, 255, 0),
                     3)

            for i in range(21):
                cv2.circle(white, (pts[i][0] + os, pts[i][1] + os1), 2, (0, 0, 255), 1)

            cv2.imshow("skeleton", white)
            # cv2.imshow("5", skeleton5)
        hands = hd.findHands(white, draw=False, flipType=True)
        if hands:
            hand = hands[0]
            x, y, w, h = hand['bbox']
            cv2.rectangle(white, (x - offset, y - offset), (x + w, y + h), (3, 255, 25), 3)

        image1 = frame[y - offset:y + h + offset, x - offset:x + w + offset]

        roi1 = image1   #rdb image with drawing

        #for gray image with drawings
        gray1 = cv2.cvtColor(roi1, cv2.COLOR_BGR2GRAY)
        blur1 = cv2.GaussianBlur(gray1, (1, 1), 2)


        test_image2= blur1
        img_final2= np.ones((400, 400), np.uint8) * 148
        h = test_image2.shape[0]
        w = test_image2.shape[1]
        img_final2[((400 - h) // 2):((400 - h) // 2) + h, ((400 - w) // 2):((400 - w) // 2) + w] = test_image2


        #cv2.imshow("aaa",white)
        # cv2.imshow("gray",img_final2)
        cv2.imshow("binary", img_final)

    cv2.imshow("frame", frame)
    interrupt = cv2.waitKey(1)
    if interrupt & 0xFF == 27:
        # esc key
        break
    if interrupt & 0xFF == ord('n'):
```

```
        if interrupt & 0xFF == ord('n'):
            p_dir = chr(ord(p_dir) + 1)
            c_dir = chr(ord(c_dir) + 1)
            if ord(p_dir)==ord('Z')+1:
                p_dir="A"
                c_dir="a"
            flag = False
            # #training data
            # count = len(os.listdir("D://sign2text_dataset_2.0/Binary_imgs//" + p_dir + "//"))

            # test data
            count = len(os.listdir("D://test_data_2.0/Gray_imgs//" + p_dir + "//"))

        if interrupt & 0xFF == ord('a'):
            if flag:
                flag=False
            else:
                suv=0
                flag=True

        print("=====",flag)
        if flag==True:

            if suv==50:
                flag=False
            if step%2==0:
                # #this is for training data collection
                # cv2.imwrite("D:\\sign2text_dataset_2.0\\Binary_imgs\\" + p_dir + "\\" + c_dir + str(count) + ".jpg", img_final)
                # cv2.imwrite("D:\\sign2text_dataset_2.0\\Gray_imgs\\" + p_dir + "\\" + c_dir + str(count) + ".jpg", img_final1)
                # cv2.imwrite("D:\\sign2text_dataset_2.0\\Gray_imgs_with_drawing\\" + p_dir + "\\" + c_dir + str(count) + ".jpg", img_final2)

                # this is for testing data collection
                # cv2.imwrite("D:\\test_data_2.0\\Binary_imgs\\" + p_dir + "\\" + c_dir + str(count) + ".jpg",
                #             img_final)
                #cv2.imwrite("D:\\test_data_2.0\\Gray_imgs\\" + p_dir + "\\" + c_dir + str(count) + ".jpg",
                            #img_final1)
                #cv2.imwrite(
                    #"D:\\test_data_2.0\\Gray_imgs_with_drawing\\" + p_dir + "\\" + c_dir + str(count) + ".jpg",
                    #img_final2)
```

```
            if suv==50:
                flag=False
            if step%2==0:
                # #this is for training data collection
                # cv2.imwrite("D:\\sign2text_dataset_2.0\\Binary_imgs\\" + p_dir + "\\" + c_dir + str(count) + ".jpg", img_final)
                # cv2.imwrite("D:\\sign2text_dataset_2.0\\Gray_imgs\\" + p_dir + "\\" + c_dir + str(count) + ".jpg", img_final1)
                # cv2.imwrite("D:\\sign2text_dataset_2.0\\Gray_imgs_with_drawing\\" + p_dir + "\\" + c_dir + str(count) + ".jpg", img_final2)

                # this is for testing data collection
                # cv2.imwrite("D:\\test_data_2.0\\Binary_imgs\\" + p_dir + "\\" + c_dir + str(count) + ".jpg",
                #             img_final)
                #cv2.imwrite("D:\\test_data_2.0\\Gray_imgs\\" + p_dir + "\\" + c_dir + str(count) + ".jpg",
                            #img_final1)
                #cv2.imwrite(
                    #"D:\\test_data_2.0\\Gray_imgs_with_drawing\\" + p_dir + "\\" + c_dir + str(count) + ".jpg",
                    #img_final2)

                count += 1
                suv += 1
            step+=1
    except Exception:
        print("==",traceback.format_exc() )

capture.release()
cv2.destroyAllWindows()
```

```
import cv2
from cvzone.HandTrackingModule import HandDetector
import numpy as np
import os as oss
import traceback



capture = cv2.VideoCapture(0)
hd = HandDetector(maxHands=1)
hd2 = HandDetector(maxHands=1)

count = len(oss.listdir("C:\\Users\\saran\\Desktop\\New folder\\titu mama\\Sign-Language-To-Text-and-Speech-Conversion\\AtoZ_3.1\\A\\"))
c_dir = 'A'

offset = 15
step = 1
flag=False
suv=0

white=np.ones((400,400),np.uint8)*255
cv2.imwrite("C:\\Users\\saran\\Desktop\\New folder\\titu mama\\Sign-Language-To-Text-and-Speech-Conversion\\white.jpg",white)
```

```
while True:
    try:
        _, frame = capture.read()
        frame = cv2.flip(frame, 1)
        hands= hd.findHands(frame, draw=False, flipType=True)
        white = cv2.imread("C:\\Users\\saran\\Desktop\\New folder\\titu mama\\Sign-Language-To-Text-and-Speech-Conversion\\white.jpg")

        if hands:
            hand = hands[0]
            x, y, w, h = hand['bbox']
            image =np.array( frame[y - offset:y + h + offset, x - offset:x + w + offset])

            handz,imz = hd2.findHands(image, draw=True, flipType=True)
            if handz:
                hand = handz[0]
                pts = hand['lmList']
                # x1,y1,w1,h1=hand['bbox']
                os=((400-w)//2)-15
                os1=((400-h)//2)-15
                for t in range(0,4,1):
                    cv2.line(white,(pts[t][0]+os,pts[t][1]+os1),(pts[t+1][0]+os,pts[t+1][1]+os1),(0,255,0),3)
                for t in range(5,8,1):
                    cv2.line(white,(pts[t][0]+os,pts[t][1]+os1),(pts[t+1][0]+os,pts[t+1][1]+os1),(0,255,0),3)
                for t in range(9,12,1):
                    cv2.line(white,(pts[t][0]+os,pts[t][1]+os1),(pts[t+1][0]+os,pts[t+1][1]+os1),(0,255,0),3)
                for t in range(13,16,1):
                    cv2.line(white,(pts[t][0]+os,pts[t][1]+os1),(pts[t+1][0]+os,pts[t+1][1]+os1),(0,255,0),3)
                for t in range(17,20,1):
                    cv2.line(white,(pts[t][0]+os,pts[t][1]+os1),(pts[t+1][0]+os,pts[t+1][1]+os1),(0,255,0),3)
                cv2.line(white, (pts[5][0]+os, pts[5][1]+os1), (pts[9][0]+os, pts[9][1]+os1), (0, 255, 0), 3)
                cv2.line(white, (pts[9][0]+os, pts[9][1]+os1), (pts[13][0]+os, pts[13][1]+os1), (0, 255, 0), 3)
                cv2.line(white, (pts[13][0]+os, pts[13][1]+os1), (pts[17][0]+os, pts[17][1]+os1), (0, 255, 0), 3)
                cv2.line(white, (pts[0][0]+os, pts[0][1]+os1), (pts[5][0]+os, pts[5][1]+os1), (0, 255, 0), 3)
                cv2.line(white, (pts[0][0]+os, pts[0][1]+os1), (pts[17][0]+os, pts[17][1]+os1), (0, 255, 0), 3)
```

```
            skeleton0=np.array(white)
            zz=np.array(white)
            for i in range(21):
                cv2.circle(white,(pts[i][0]+os,pts[i][1]+os1),2,(0 , 0 , 255),1)

            skeleton1=np.array(white)

            cv2.imshow("1",skeleton1)

        frame = cv2.putText(frame, "dir=" + str(c_dir) + "  count=" + str(count), (50,50),
                            cv2.FONT_HERSHEY_SIMPLEX,
                            1, (255, 0, 0), 1, cv2.LINE_AA)
        cv2.imshow("frame", frame)
        interrupt = cv2.waitKey(1)
        if interrupt & 0xFF == 27:
            # esc key
            break


        if interrupt & 0xFF == ord('n'):
            c_dir = chr(ord(c_dir)+1)
            if ord(c_dir)==ord('Z')+1:
                c_dir='A'
            flag = False
            count = len(oss.listdir("C:\\Users\\saran\\Desktop\\New folder\\titu mama\\Sign-Language-To-Text-and-Speech-Conversion\\AtoZ_3.1\\A\\" + (c_dir) + "\\"))

        if interrupt & 0xFF == ord('a'):
            if flag:
                flag=False
            else:
                suv=0
                flag=True
```

```
        print("=====",flag)
        if flag==True:

            if suv==180:
                flag=False
            if step%3==0:
                cv2.imwrite("C:\\Users\\saran\\Desktop\\New folder\\titu mama\\Sign-Language-To-Text-and-Speech-Conversion\\AtoZ_3.1\\A\\" + (c_dir) + "\\" + str(count) + ".
                            skeleton1)

                count += 1
                suv += 1
            step+=1



    except Exception:
        print("==",traceback.format_exc() )

capture.release()
cv2.destroyAllWindows()
```

Python

```
# Importing Libraries
import numpy as np
import math
import cv2
import os, sys
import traceback
import pyttsx3
from keras.models import load_model
from cvzone.HandTrackingModule import HandDetector
from string import ascii_uppercase
hd = HandDetector(maxHands=1)
hd2 = HandDetector(maxHands=1)
import tkinter as tk
from PIL import Image, ImageTk

offset=29

os.environ["THEANO_FLAGS"] = "device=cuda, assert_no_cpu_op=True"
```

```
# Application :

class Application:

    def __init__(self):
        self.vs = cv2.VideoCapture(0)
        self.current_image = None
        self.model = load_model("C:\\Users\\saran\\Desktop\\New folder\\titu mama\\Sign-Language-To-Text-and-Speech-Conversion\\cnn8grps_rad1_model.h5")
        self.speak_engine=pyttsx3.init()
        self.speak_engine.setProperty("rate",100)
        voices=self.speak_engine.getProperty("voices")
        self.speak_engine.setProperty("voice",voices[0].id)

        self.ct = {}
        self.ct['blank'] = 0
        self.blank_flag = 0
        self.space_flag=False
        self.next_flag=True
        self.prev_char=""
        self.count=-1
        self.ten_prev_char=[]
        for i in range(10):
            self.ten_prev_char.append(" ")


        for i in ascii_uppercase:
            self.ct[i] = 0
```

```
print("Loaded model from disk")

self.root = tk.Tk()
self.root.title("Sign Language To Text Conversion")
self.root.protocol('WM_DELETE_WINDOW', self.destructor)
self.root.geometry("1300x700")

self.panel = tk.Label(self.root)
self.panel.place(x=100, y=3, width=480, height=640)

self.panel2 = tk.Label(self.root)  # initialize image panel
self.panel2.place(x=700, y=115, width=400, height=400)

self.T = tk.Label(self.root)
self.T.place(x=60, y=5)
self.T.config(text="Sign Language To Text Conversion", font=("Courier", 30, "bold"))

self.panel3 = tk.Label(self.root)  # Current Symbol
self.panel3.place(x=280, y=585)

self.T1 = tk.Label(self.root)
self.T1.place(x=10, y=580)
self.T1.config(text="Character :", font=("Courier", 30, "bold"))

self.panel5 = tk.Label(self.root)  # Sentence
self.panel5.place(x=260, y=632)

self.T3 = tk.Label(self.root)
self.T3.place(x=10, y=632)
self.T3.config(text="Sentence :", font=("Courier", 30, "bold"))

self.T4 = tk.Label(self.root)
self.T4.place(x=10, y=700)
self.T4.config(text="Suggestions :", fg="red", font=("Courier", 30, "bold"))
```

```
self.b1=tk.Button(self.root)
self.b1.place(x=390,y=700)

self.b2 = tk.Button(self.root)
self.b2.place(x=590, y=700)

self.b3 = tk.Button(self.root)
self.b3.place(x=790, y=700)

self.b4 = tk.Button(self.root)
self.b4.place(x=990, y=700)

self.speak = tk.Button(self.root)
self.speak.place(x=1305, y=630)
self.speak.config(text="Speak", font=("Courier", 20), wraplength=100, command=self.speak_fun)

self.clear = tk.Button(self.root)
self.clear.place(x=1205, y=630)
self.clear.config(text="Clear", font=("Courier", 20), wraplength=100, command=self.clear_fun)
```

```
self.str = " "
self.ccc=0
self.word = " "
self.current_symbol = "C"
self.photo = "Empty"


self.word1=" "
self.word2 = " "
self.word3 = " "
self.word4 = " "

self.video_loop()
```

```
def video_loop(self):
    try:
        ok, frame = self.vs.read()
        cv2image = cv2.flip(frame, 1)
        hands = hd.findHands(cv2image, draw=False, flipType=True)
        cv2image_copy=np.array(cv2image)
        cv2image = cv2.cvtColor(cv2image, cv2.COLOR_BGR2RGB)
        self.current_image = Image.fromarray(cv2image)
        imgtk = ImageTk.PhotoImage(image=self.current_image)
        self.panel.imgtk = imgtk
        self.panel.config(image=imgtk)

        if hands:
            # #print(" --------- lmlist=",hands[1])
            hand = hands[0]
            x, y, w, h = hand['bbox']
            image = cv2image_copy[y - offset:y + h + offset, x - offset:x + w + offset]

            white = cv2.imread("C:\\Users\\saran\\Desktop\\New folder\\titu mama\\Sign-Language-To-Text-and-Speech-Conversion\\white.jpg")
            # img_final=img_final1=img_final2=0

            handz = hd2.findHands(image, draw=False, flipType=True)
            print(" ", self.ccc)
            self.ccc += 1
```

```
    if handz:
        hand = handz[0]
        self.pts = hand['lmList']
        # x1,y1,w1,h1=hand['bbox']

        os = ((400 - w) // 2) - 15
        os1 = ((400 - h) // 2) - 15
        for t in range(0, 4, 1):
            cv2.line(white, (self.pts[t][0] + os, self.pts[t][1] + os1), (self.pts[t + 1][0] + os, self.pts[t + 1][1] + os1),
                     (0, 255, 0), 3)
        for t in range(5, 8, 1):
            cv2.line(white, (self.pts[t][0] + os, self.pts[t][1] + os1), (self.pts[t + 1][0] + os, self.pts[t + 1][1] + os1),
                     (0, 255, 0), 3)
        for t in range(9, 12, 1):
            cv2.line(white, (self.pts[t][0] + os, self.pts[t][1] + os1), (self.pts[t + 1][0] + os, self.pts[t + 1][1] + os1),
                     (0, 255, 0), 3)
        for t in range(13, 16, 1):
            cv2.line(white, (self.pts[t][0] + os, self.pts[t][1] + os1), (self.pts[t + 1][0] + os, self.pts[t + 1][1] + os1),
                     (0, 255, 0), 3)
        for t in range(17, 20, 1):
            cv2.line(white, (self.pts[t][0] + os, self.pts[t][1] + os1), (self.pts[t + 1][0] + os, self.pts[t + 1][1] + os1),
                     (0, 255, 0), 3)
        cv2.line(white, (self.pts[5][0] + os, self.pts[5][1] + os1), (self.pts[9][0] + os, self.pts[9][1] + os1), (0, 255, 0),
                 3)
        cv2.line(white, (self.pts[9][0] + os, self.pts[9][1] + os1), (self.pts[13][0] + os, self.pts[13][1] + os1), (0, 255, 0),
                 3)
        cv2.line(white, (self.pts[13][0] + os, self.pts[13][1] + os1), (self.pts[17][0] + os, self.pts[17][1] + os1),
                 (0, 255, 0), 3)
        cv2.line(white, (self.pts[0][0] + os, self.pts[0][1] + os1), (self.pts[5][0] + os, self.pts[5][1] + os1), (0, 255, 0),
                 3)
        cv2.line(white, (self.pts[0][0] + os, self.pts[0][1] + os1), (self.pts[17][0] + os, self.pts[17][1] + os1), (0, 255, 0),
                 3)
```

```
for i in range(21):
    cv2.circle(white, (self.pts[i][0] + os, self.pts[i][1] + os1), 2, (0, 0, 255), 1)

res=white
self.predict(res)

self.current_image2 = Image.fromarray(res)

imgtk = ImageTk.PhotoImage(image=self.current_image2)

self.panel2.imgtk = imgtk
self.panel2.config(image=imgtk)

self.panel3.config(text=self.current_symbol, font=("Courier", 30))

#self.panel4.config(text=self.word, font=("Courier", 30))



self.b1.config(text=self.word1, font=("Courier", 20), wraplength=825, command=self.action1)
self.b2.config(text=self.word2, font=("Courier", 20), wraplength=825,  command=self.action2)
self.b3.config(text=self.word3, font=("Courier", 20), wraplength=825,  command=self.action3)
self.b4.config(text=self.word4, font=("Courier", 20), wraplength=825,  command=self.action4)
```

```
        self.panel5.config(text=self.str, font=("Courier", 30), wraplength=1025)
    except Exception:
        print("==", traceback.format_exc())
    finally:
        self.root.after(1, self.video_loop)

def distance(self,x,y):
    return math.sqrt(((x[0] - y[0]) ** 2) + ((x[1] - y[1]) ** 2))

def action1(self):
    idx_space = self.str.rfind(" ")
    idx_word = self.str.find(self.word, idx_space)
    last_idx = len(self.str)
    self.str = self.str[:idx_word]
    self.str = self.str + self.word1.upper()
```

```
def action2(self):
    idx_space = self.str.rfind(" ")
    idx_word = self.str.find(self.word, idx_space)
    last_idx = len(self.str)
    self.str=self.str[:idx_word]
    self.str=self.str+self.word2.upper()
    #self.str[idx_word:last_idx] = self.word2


def action3(self):
    idx_space = self.str.rfind(" ")
    idx_word = self.str.find(self.word, idx_space)
    last_idx = len(self.str)
    self.str = self.str[:idx_word]
    self.str = self.str + self.word3.upper()



def action4(self):
    idx_space = self.str.rfind(" ")
    idx_word = self.str.find(self.word, idx_space)
    last_idx = len(self.str)
    self.str = self.str[:idx_word]
    self.str = self.str + self.word4.upper()


def speak_fun(self):
    self.speak_engine.say(self.str)
    self.speak_engine.runAndWait()
```

```
def clear_fun(self):
    self.str=" "
    self.word1 = " "
    self.word2 = " "
    self.word3 = " "
    self.word4 = " "

def predict(self, test_image):
    white=test_image
    white = white.reshape(1, 400, 400, 3)
    prob = np.array(self.model.predict(white)[0], dtype='float32')
    ch1 = np.argmax(prob, axis=0)
    prob[ch1] = 0
    ch2 = np.argmax(prob, axis=0)
    prob[ch2] = 0
    ch3 = np.argmax(prob, axis=0)
    prob[ch3] = 0

    pl = [ch1, ch2]
```

```
# condition for [Aemnst]
l = [[5, 2], [5, 3], [3, 5], [3, 6], [3, 0], [3, 2], [6, 4], [6, 1], [6, 2], [6, 6], [6, 7], [6, 0], [6, 5],
     [4, 1], [1, 0], [1, 1], [6, 3], [1, 6], [5, 6], [5, 1], [4, 5], [1, 4], [1, 5], [2, 0], [2, 6], [4, 6],
     [1, 0], [5, 7], [1, 6], [6, 1], [7, 6], [2, 5], [7, 1], [5, 4], [7, 0], [7, 5], [7, 2]]
if pl in l:
    if (self.pts[6][1] < self.pts[8][1] and self.pts[10][1] < self.pts[12][1] and self.pts[14][1] < self.pts[16][1] and self.pts[18][1] < self.pts[20][
        1]):
        ch1 = 0
        # print("00000")

# condition for [o][s]
l = [[2, 2], [2, 1]]
if pl in l:
    if (self.pts[5][0] < self.pts[4][0]):
        ch1 = 0
        print("++++++++++++++++++")
        # print("00000")

# condition for [c0][aemnst]
l = [[0, 0], [0, 6], [0, 2], [0, 5], [0, 1], [0, 7], [5, 2], [7, 6], [7, 1]]
pl = [ch1, ch2]
if pl in l:
    if (self.pts[0][0] > self.pts[8][0] and self.pts[0][0] > self.pts[4][0] and self.pts[0][0] > self.pts[12][0] and self.pts[0][0] > self.pts[16][
        0] and self.pts[0][0] > self.pts[20][0]) and self.pts[5][0] > self.pts[4][0]:
        ch1 = 2
        # print("22222")
```

```
# condition for [c0][aemnst]
l = [[6, 0], [6, 6], [6, 2]]
pl = [ch1, ch2]
if pl in l:
    if self.distance(self.pts[8], self.pts[16]) < 52:
        ch1 = 2
        # print("22222")


# condition for [gh][bdfikruvw]
l = [[1, 4], [1, 5], [1, 6], [1, 3], [1, 0]]
pl = [ch1, ch2]

if pl in l:
    if self.pts[6][1] > self.pts[8][1] and self.pts[14][1] < self.pts[16][1] and self.pts[18][1] < self.pts[20][1] and self.pts[0][0] < self.pts[8][
        0] and self.pts[0][0] < self.pts[12][0] and self.pts[0][0] < self.pts[16][0] and self.pts[0][0] < self.pts[20][0]:
        ch1 = 3
        print("33333c")
```

```
# con for [gh][l]
l = [[4, 6], [4, 1], [4, 5], [4, 3], [4, 7]]
pl = [ch1, ch2]
if pl in l:
    if self.pts[4][0] > self.pts[0][0]:
        ch1 = 3
        print("33333b")

# con for [gh][pqz]
l = [[5, 3], [5, 0], [5, 7], [5, 4], [5, 2], [5, 1], [5, 5]]
pl = [ch1, ch2]
if pl in l:
    if self.pts[2][1] + 15 < self.pts[16][1]:
        ch1 = 3
        print("33333a")

# con for [l][x]
l = [[6, 4], [6, 1], [6, 2]]
pl = [ch1, ch2]
if pl in l:
    if self.distance(self.pts[4], self.pts[11]) > 55:
        ch1 = 4
        # print("44444")

# con for [l][d]
l = [[1, 4], [1, 6], [1, 1]]
pl = [ch1, ch2]
if pl in l:
    if (self.distance(self.pts[4], self.pts[11]) > 50) and (
            self.pts[6][1] > self.pts[8][1] and self.pts[10][1] < self.pts[12][1] and self.pts[14][1] < self.pts[16][1] and self.pts[18][1] <
            self.pts[20][1]):
        ch1 = 4
        # print("44444")
```

```
# con for [l][gh]
l = [[3, 6], [3, 4]]
pl = [ch1, ch2]
if pl in l:
    if (self.pts[4][0] < self.pts[0][0]):
        ch1 = 4
        # print("44444")

# con for [l][c0]
l = [[2, 2], [2, 5], [2, 4]]
pl = [ch1, ch2]
if pl in l:
    if (self.pts[1][0] < self.pts[12][0]):
        ch1 = 4
        # print("44444")

# con for [l][c0]
l = [[2, 2], [2, 5], [2, 4]]
pl = [ch1, ch2]
if pl in l:
    if (self.pts[1][0] < self.pts[12][0]):
        ch1 = 4
        # print("44444")
```

```
# con for [gh][z]
l = [[3, 6], [3, 5], [3, 4]]
pl = [ch1, ch2]
if pl in l:
    if (self.pts[6][1] > self.pts[8][1] and self.pts[10][1] < self.pts[12][1] and self.pts[14][1] < self.pts[16][1] and self.pts[18][1] < self.pts[20][
        1]) and self.pts[4][1] > self.pts[10][1]:
        ch1 = 5
        print("55555b")

# con for [gh][pq]
l = [[3, 2], [3, 1], [3, 6]]
pl = [ch1, ch2]
if pl in l:
    if self.pts[4][1] + 17 > self.pts[8][1] and self.pts[4][1] + 17 > self.pts[12][1] and self.pts[4][1] + 17 > self.pts[16][1] and self.pts[4][
        1] + 17 > self.pts[20][1]:
        ch1 = 5
        print("55555a")

# con for [l][pqz]
l = [[4, 4], [4, 5], [4, 2], [7, 5], [7, 6], [7, 0]]
pl = [ch1, ch2]
if pl in l:
    if self.pts[4][0] > self.pts[0][0]:
        ch1 = 5
        # print("55555")
```

```
# con for [pqz][aemnst]
l = [[0, 2], [0, 6], [0, 1], [0, 5], [0, 0], [0, 7], [0, 4], [0, 3], [2, 7]]
pl = [ch1, ch2]
if pl in l:
    if self.pts[0][0] < self.pts[8][0] and self.pts[0][0] < self.pts[12][0] and self.pts[0][0] < self.pts[16][0] and self.pts[0][0] < self.pts[20][0]:
        ch1 = 5
        # print("55555")

# con for [pqz][yj]
l = [[5, 7], [5, 2], [5, 6]]
pl = [ch1, ch2]
if pl in l:
    if self.pts[3][0] < self.pts[0][0]:
        ch1 = 7
        # print("77777")

# con for [l][yj]
l = [[4, 6], [4, 2], [4, 4], [4, 1], [4, 5], [4, 7]]
pl = [ch1, ch2]
if pl in l:
    if self.pts[6][1] < self.pts[8][1]:
        ch1 = 7
        # print("77777")
```

```
# con for [x][yj]
l = [[6, 7], [0, 7], [0, 1], [0, 0], [6, 4], [6, 6], [6, 5], [6, 1]]
pl = [ch1, ch2]
if pl in l:
    if self.pts[18][1] > self.pts[20][1]:
        ch1 = 7
        # print("77777")

# condition for [x][aemnst]
l = [[0, 4], [0, 2], [0, 3], [0, 1], [0, 6]]
pl = [ch1, ch2]
if pl in l:
    if self.pts[5][0] > self.pts[16][0]:
        ch1 = 6
        print("666661")


# condition for [yj][x]
print("2222  ch1=+++++++++++++++++", ch1, ",", ch2)
l = [[7, 2]]
pl = [ch1, ch2]
if pl in l:
    if self.pts[18][1] < self.pts[20][1] and self.pts[8][1] < self.pts[10][1]:
        ch1 = 6
        print("666662")
```

```
# condition for [c0][x]
l = [[2, 1], [2, 2], [2, 6], [2, 7], [2, 0]]
pl = [ch1, ch2]
if pl in l:
    if self.distance(self.pts[8], self.pts[16]) > 50:
        ch1 = 6
        print("666663")

# con for [l][x]

l = [[4, 6], [4, 2], [4, 1], [4, 4]]
pl = [ch1, ch2]
if pl in l:
    if self.distance(self.pts[4], self.pts[11]) < 60:
        ch1 = 6
        print("666664")

# con for [x][d]
l = [[1, 4], [1, 6], [1, 0], [1, 2]]
pl = [ch1, ch2]
if pl in l:
    if self.pts[5][0] - self.pts[4][0] - 15 > 0:
        ch1 = 6
        print("666665")
```

```
# con for [b][pqz]
l = [[5, 0], [5, 1], [5, 4], [5, 5], [5, 6], [6, 1], [7, 6], [0, 2], [7, 1], [7, 4], [6, 6], [7, 2], [5, 0],
     [6, 3], [6, 4], [7, 5], [7, 2]]
pl = [ch1, ch2]
if pl in l:
    if (self.pts[6][1] > self.pts[8][1] and self.pts[10][1] > self.pts[12][1] and self.pts[14][1] > self.pts[16][1] and self.pts[18][1] > self.pts[20][
        1]):
        ch1 = 1
        print("111111")

# con for [f][pqz]
l = [[6, 1], [6, 0], [0, 3], [6, 4], [2, 2], [0, 6], [6, 2], [7, 6], [4, 6], [4, 1], [4, 2], [0, 2], [7, 1],
     [7, 4], [6, 6], [7, 2], [7, 5], [7, 2]]
pl = [ch1, ch2]
if pl in l:
    if (self.pts[6][1] < self.pts[8][1] and self.pts[10][1] > self.pts[12][1] and self.pts[14][1] > self.pts[16][1] and
            self.pts[18][1] > self.pts[20][1]):
        ch1 = 1
        print("111112")

l = [[6, 1], [6, 0], [4, 2], [4, 1], [4, 6], [4, 4]]
pl = [ch1, ch2]
if pl in l:
    if (self.pts[10][1] > self.pts[12][1] and self.pts[14][1] > self.pts[16][1] and
            self.pts[18][1] > self.pts[20][1]):
        ch1 = 1
        print("111112")
```

```
# con for [d][pqz]
fg = 19
# print("_________________ch1=",ch1," ch2=",ch2)
l = [[5, 0], [3, 4], [3, 0], [3, 1], [3, 5], [5, 5], [5, 4], [5, 1], [7, 6]]
pl = [ch1, ch2]
if pl in l:
    if ((self.pts[6][1] > self.pts[8][1] and self.pts[10][1] < self.pts[12][1] and self.pts[14][1] < self.pts[16][1] and
         self.pts[18][1] < self.pts[20][1]) and (self.pts[2][0] < self.pts[0][0]) and self.pts[4][1] > self.pts[14][1]):
        ch1 = 1
        print("111113")

l = [[4, 1], [4, 2], [4, 4]]
pl = [ch1, ch2]
if pl in l:
    if (self.distance(self.pts[4], self.pts[11]) < 50) and (
            self.pts[6][1] > self.pts[8][1] and self.pts[10][1] < self.pts[12][1] and self.pts[14][1] < self.pts[16][1] and self.pts[18][1] <
            self.pts[20][1]):
        ch1 = 1
        print("1111993")

l = [[3, 4], [3, 0], [3, 1], [3, 5], [3, 6]]
pl = [ch1, ch2]
if pl in l:
    if ((self.pts[6][1] > self.pts[8][1] and self.pts[10][1] < self.pts[12][1] and self.pts[14][1] < self.pts[16][1] and
         self.pts[18][1] < self.pts[20][1]) and (self.pts[2][0] < self.pts[0][0]) and self.pts[14][1] < self.pts[4][1]):
        ch1 = 1
        print("1111mmm3")

l = [[6, 6], [6, 4], [6, 1], [6, 2]]
pl = [ch1, ch2]
if pl in l:
    if self.pts[5][0] - self.pts[4][0] - 15 < 0:
        ch1 = 1
        print("1111140")
```

```
# con for [i][pqz]
l = [[5, 4], [5, 5], [5, 1], [0, 3], [0, 7], [5, 0], [0, 2], [6, 2], [7, 5], [7, 1], [7, 6], [7, 7]]
pl = [ch1, ch2]
if pl in l:
    if ((self.pts[6][1] < self.pts[8][1] and self.pts[10][1] < self.pts[12][1] and self.pts[14][1] < self.pts[16][1] and
         self.pts[18][1] > self.pts[20][1])):
        ch1 = 1
        print("111114")

# con for [yj][bfdi]
l = [[1, 5], [1, 7], [1, 1], [1, 6], [1, 3], [1, 0]]
pl = [ch1, ch2]
if pl in l:
    if (self.pts[4][0] < self.pts[5][0] + 15) and (
    (self.pts[6][1] < self.pts[8][1] and self.pts[10][1] < self.pts[12][1] and self.pts[14][1] < self.pts[16][1] and
     self.pts[18][1] > self.pts[20][1])):
        ch1 = 7
        print("111114lll;;p")

# con for [uvr]
l = [[5, 5], [5, 0], [5, 4], [5, 1], [4, 6], [4, 1], [7, 6], [3, 0], [3, 5]]
pl = [ch1, ch2]
if pl in l:
    if ((self.pts[6][1] > self.pts[8][1] and self.pts[10][1] > self.pts[12][1] and self.pts[14][1] < self.pts[16][1] and
         self.pts[18][1] < self.pts[20][1])) and self.pts[4][1] > self.pts[14][1]:
        ch1 = 1
        print("111115")
```

```
# con for [w]
fg = 13
l = [[3, 5], [3, 0], [3, 6], [5, 1], [4, 1], [2, 0], [5, 0], [5, 5]]
pl = [ch1, ch2]
if pl in l:
    if not (self.pts[0][0] + fg < self.pts[8][0] and self.pts[0][0] + fg < self.pts[12][0] and self.pts[0][0] + fg < self.pts[16][0] and
            self.pts[0][0] + fg < self.pts[20][0]) and not (
            self.pts[0][0] > self.pts[8][0] and self.pts[0][0] > self.pts[12][0] and self.pts[0][0] > self.pts[16][0] and self.pts[0][0] > self.pts[20][
        0]) and self.distance(self.pts[4], self.pts[11]) < 50:
        ch1 = 1
        print("111116")

# con for [w]

l = [[5, 0], [5, 5], [0, 1]]
pl = [ch1, ch2]
if pl in l:
    if self.pts[6][1] > self.pts[8][1] and self.pts[10][1] > self.pts[12][1] and self.pts[14][1] > self.pts[16][1]:
        ch1 = 1
        print("1117")

# -------------------------condn for 8 groups  ends

# -------------------------condn for subgroups  starts
```

```
if ch1 == 0:
    ch1 = 'S'
    if self.pts[4][0] < self.pts[6][0] and self.pts[4][0] < self.pts[10][0] and self.pts[4][0] < self.pts[14][0] and self.pts[4][0] < self.pts[18][0]:
        ch1 = 'A'
    if self.pts[4][0] > self.pts[6][0] and self.pts[4][0] < self.pts[10][0] and self.pts[4][0] < self.pts[14][0] and self.pts[4][0] < self.pts[18][
        0] and self.pts[4][1] < self.pts[14][1] and self.pts[4][1] < self.pts[18][1]:
        ch1 = 'T'
    if self.pts[4][1] > self.pts[8][1] and self.pts[4][1] > self.pts[12][1] and self.pts[4][1] > self.pts[16][1] and self.pts[4][1] > self.pts[20][1]:
        ch1 = 'E'
    if self.pts[4][0] > self.pts[6][0] and self.pts[4][0] > self.pts[10][0] and self.pts[4][0] > self.pts[14][0] and self.pts[4][1] < self.pts[18][1]:
        ch1 = 'M'
    if self.pts[4][0] > self.pts[6][0] and self.pts[4][0] > self.pts[10][0] and self.pts[4][1] < self.pts[18][1] and self.pts[4][1] < self.pts[14][1]:
        ch1 = 'N'
```

```
if ch1 == 2:
    if self.distance(self.pts[12], self.pts[4]) > 42:
        ch1 = 'C'
    else:
        ch1 = 'O'

if ch1 == 3:
    if (self.distance(self.pts[8], self.pts[12])) > 72:
        ch1 = 'G'
    else:
        ch1 = 'H'

if ch1 == 7:
    if self.distance(self.pts[8], self.pts[4]) > 42:
        ch1 = 'Y'
    else:
        ch1 = 'J'

if ch1 == 4:
    ch1 = 'L'

if ch1 == 6:
    ch1 = 'X'

if ch1 == 5:
    if self.pts[4][0] > self.pts[12][0] and self.pts[4][0] > self.pts[16][0] and self.pts[4][0] > self.pts[20][0]:
        if self.pts[8][1] < self.pts[5][1]:
            ch1 = 'Z'
        else:
            ch1 = 'Q'
    else:
        ch1 = 'P'
```

```
if ch1 == 1:
    if (self.pts[6][1] > self.pts[8][1] and self.pts[10][1] > self.pts[12][1] and self.pts[14][1] > self.pts[16][1] and self.pts[18][1] > self.pts[20][
        1]):
        ch1 = 'B'
    if (self.pts[6][1] > self.pts[8][1] and self.pts[10][1] < self.pts[12][1] and self.pts[14][1] < self.pts[16][1] and self.pts[18][1] < self.pts[20][
        1]):
        ch1 = 'D'
    if (self.pts[6][1] < self.pts[8][1] and self.pts[10][1] > self.pts[12][1] and self.pts[14][1] > self.pts[16][1] and self.pts[18][1] > self.pts[20][
        1]):
        ch1 = 'F'
    if (self.pts[6][1] < self.pts[8][1] and self.pts[10][1] < self.pts[12][1] and self.pts[14][1] < self.pts[16][1] and self.pts[18][1] > self.pts[20][
        1]):
        ch1 = 'I'
    if (self.pts[6][1] > self.pts[8][1] and self.pts[10][1] > self.pts[12][1] and self.pts[14][1] > self.pts[16][1] and self.pts[18][1] < self.pts[20][
        1]):
        ch1 = 'W'
    if (self.pts[6][1] > self.pts[8][1] and self.pts[10][1] > self.pts[12][1] and self.pts[14][1] < self.pts[16][1] and self.pts[18][1] < self.pts[20][
        1]) and self.pts[4][1] < self.pts[9][1]:
        ch1 = 'K'
    if ((self.distance(self.pts[8], self.pts[12]) - self.distance(self.pts[6], self.pts[10])) < 8) and (
            self.pts[6][1] > self.pts[8][1] and self.pts[10][1] > self.pts[12][1] and self.pts[14][1] < self.pts[16][1] and self.pts[18][1] <
            self.pts[20][1]):
        ch1 = 'U'
    if ((self.distance(self.pts[8], self.pts[12]) - self.distance(self.pts[6], self.pts[10])) >= 8) and (
            self.pts[6][1] > self.pts[8][1] and self.pts[10][1] > self.pts[12][1] and self.pts[14][1] < self.pts[16][1] and self.pts[18][1] <
            self.pts[20][1]) and (self.pts[4][1] > self.pts[9][1]):
        ch1 = 'V'

    if (self.pts[8][0] > self.pts[12][0]) and (
            self.pts[6][1] > self.pts[8][1] and self.pts[10][1] > self.pts[12][1] and self.pts[14][1] < self.pts[16][1] and self.pts[18][1] <
            self.pts[20][1]):
        ch1 = 'R'

if ch1 == 1 or ch1 =='E' or ch1 =='S' or ch1 =='X' or ch1 =='Y' or ch1 =='B':
    if (self.pts[6][1] > self.pts[8][1] and self.pts[10][1] < self.pts[12][1] and self.pts[14][1] < self.pts[16][1] and self.pts[18][1] > self.pts[20][1]):
        ch1=" "
```

```
print(self.pts[4][0] < self.pts[5][0])
if ch1 == 'E' or ch1=='Y' or ch1=='B':
    if (self.pts[4][0] < self.pts[5][0]) and (self.pts[6][1] > self.pts[8][1] and self.pts[10][1] > self.pts[12][1] and self.pts[14][1] > self.pts[16][1] and self.pt
        ch1="next"


if ch1 == 'Next' or 'B' or 'C' or 'H' or 'F' or 'X':
    if (self.pts[0][0] > self.pts[8][0] and self.pts[0][0] > self.pts[12][0] and self.pts[0][0] > self.pts[16][0] and self.pts[0][0] > self.pts[20][0]) and (self.pts
        ch1 = 'Backspace'


if ch1=="next" and self.prev_char!="next":
    if self.ten_prev_char[(self.count-2)%10]!="next":
        if self.ten_prev_char[(self.count-2)%10]=="Backspace":
            self.str=self.str[0:-1]
        else:
            if self.ten_prev_char[(self.count - 2) % 10] != "Backspace":
                self.str = self.str + self.ten_prev_char[(self.count-2)%10]
    else:
        if self.ten_prev_char[(self.count - 0) % 10] != "Backspace":
            self.str = self.str + self.ten_prev_char[(self.count - 0) % 10]
```

```
if ch1=="  " and self.prev_char!="  ":
    self.str = self.str + "  "

self.prev_char=ch1
self.current_symbol=ch1
self.count += 1
self.ten_prev_char[self.count%10]=ch1


if len(self.str.strip())!=0:
    st=self.str.rfind(" ")
    ed=len(self.str)
    word=self.str[st+1:ed]
    self.word=word
    print("----------word = ",word)
    if len(word.strip())!=0:
        ddd.check(word)
        lenn = len(ddd.suggest(word))
        if lenn >= 4:
            self.word4 = ddd.suggest(word)[3]

        if lenn >= 3:
            self.word3 = ddd.suggest(word)[2]

        if lenn >= 2:
            self.word2 = ddd.suggest(word)[1]

        if lenn >= 1:
            self.word1 = ddd.suggest(word)[0]
    else:
        self.word1 = " "
        self.word2 = " "
        self.word3 = " "
        self.word4 = " "
```

```
    def destructor(self):

        print("Closing Application...")
        print(self.ten_prev_char)
        self.root.destroy()
        self.vs.release()
        cv2.destroyAllWindows()


print("Starting Application...")

(Application()).root.mainloop()
```

```
import math
import cv2
from cvzone.HandTrackingModule import HandDetector
import numpy as np
from keras.models import load_model
import traceback

model = load_model('C:\\Users\\saran\\Desktop\\New folder\\titu mama\\Sign-Language-To-Text-and-Speech-Conversion\\cnn8grps_rad1_model.h5')
white = np.ones((400, 400), np.uint8) * 255
cv2.imwrite("C:\\Users\\saran\\Desktop\\New folder\\titu mama\\Sign-Language-To-Text-and-Speech-Conversion\\white.jpg", white)

capture = cv2.VideoCapture(0)

hd = HandDetector(maxHands=1)
hd2 = HandDetector(maxHands=1)

offset = 29
step = 1
flag = False
suv = 0
```

```
def distance(x, y):
    return math.sqrt(((x[0] - y[0]) ** 2) + ((x[1] - y[1]) ** 2))


def distance_3d(x, y):
    return math.sqrt(((x[0] - y[0]) ** 2) + ((x[1] - y[1]) ** 2) + ((x[2] - y[2]) ** 2))


bfh = 0
dicttt=dict()
count=0
kok=[]
```

```
while True:
    try:
        _, frame = capture.read()
        frame = cv2.flip(frame, 1)
        hands = hd.findHands(frame, draw=False, flipType=True)
        print(frame.shape)
        if hands:
            # #print(" --------- lmlist=",hands[1])
            hand = hands[0]
            x, y, w, h = hand['bbox']
            image = frame[y - offset:y + h + offset, x - offset:x + w + offset]
            white = cv2.imread("C:\\Users\\saran\\Desktop\\New folder\\titu mama\\Sign-Language-To-Text-and-Speech-Conversion\\white.jpg")
            # img_final=img_final1=img_final2=0
            handz = hd2.findHands(image, draw=False, flipType=True)
            if handz:
                hand = handz[0]
                pts = hand['lmList']
                # x1,y1,w1,h1=hand['bbox']
```

```
os = ((400 - w) // 2) - 15
os1 = ((400 - h) // 2) - 15
for t in range(0, 4, 1):
    cv2.line(white, (pts[t][0] + os, pts[t][1] + os1), (pts[t + 1][0] + os, pts[t + 1][1] + os1),
             (0, 255, 0), 3)
for t in range(5, 8, 1):
    cv2.line(white, (pts[t][0] + os, pts[t][1] + os1), (pts[t + 1][0] + os, pts[t + 1][1] + os1),
             (0, 255, 0), 3)
for t in range(9, 12, 1):
    cv2.line(white, (pts[t][0] + os, pts[t][1] + os1), (pts[t + 1][0] + os, pts[t + 1][1] + os1),
             (0, 255, 0), 3)
for t in range(13, 16, 1):
    cv2.line(white, (pts[t][0] + os, pts[t][1] + os1), (pts[t + 1][0] + os, pts[t + 1][1] + os1),
             (0, 255, 0), 3)
for t in range(17, 20, 1):
    cv2.line(white, (pts[t][0] + os, pts[t][1] + os1), (pts[t + 1][0] + os, pts[t + 1][1] + os1),
             (0, 255, 0), 3)
cv2.line(white, (pts[5][0] + os, pts[5][1] + os1), (pts[9][0] + os, pts[9][1] + os1), (0, 255, 0),
         3)
cv2.line(white, (pts[9][0] + os, pts[9][1] + os1), (pts[13][0] + os, pts[13][1] + os1), (0, 255, 0),
         3)
cv2.line(white, (pts[13][0] + os, pts[13][1] + os1), (pts[17][0] + os, pts[17][1] + os1),
         (0, 255, 0), 3)
cv2.line(white, (pts[0][0] + os, pts[0][1] + os1), (pts[5][0] + os, pts[5][1] + os1), (0, 255, 0),
         3)
cv2.line(white, (pts[0][0] + os, pts[0][1] + os1), (pts[17][0] + os, pts[17][1] + os1), (0, 255, 0),
         3)
```

```
for i in range(21):
    cv2.circle(white, (pts[i][0] + os, pts[i][1] + os1), 2, (0, 0, 255), 1)

cv2.imshow("2", white)
# cv2.imshow("5", skeleton5)

# #print(model.predict(img))
white = white.reshape(1, 400, 400, 3)
prob = np.array(model.predict(white)[0], dtype='float32')
ch1 = np.argmax(prob, axis=0)
prob[ch1] = 0
ch2 = np.argmax(prob, axis=0)
prob[ch2] = 0
ch3 = np.argmax(prob, axis=0)
prob[ch3] = 0
```

```
pl = [ch1, ch2]

#condition for [Aemnst]
l=[[5,2],[5,3],[3,5],[3,6],[3,0],[3,2],[6,4],[6,1],[6,2],[6,6],[6,7],[6,0],[6,5],[4,1],[1,0],[1,1],[6,3],[1,6],[5,6],[5,1],[4,5],[1,4],[1,5],[2,0],[2,6],[4,6
if pl in l:
    if (pts[6][1] < pts[8][1] and pts[10][1] < pts[12][1] and pts[14][1] < pts[16][1] and pts[18][1] <pts[20][1]):
        ch1=0
        #print("00000")

#condition for [o][s]
l=[[2,2],[2,1]]
if pl in l:
    if (pts[5][0] < pts[4][0] ):
        ch1=0
        print("++++++++++++++++++")
        #print("00000")



#condition for [c0][aemnst]
l=[[0,0],[0,6],[0,2],[0,5],[0,1],[0,7],[5,2],[7,6],[7,1]]
pl=[ch1,ch2]
if pl in l:
    if (pts[0][0]>pts[8][0] and pts[0][0]>pts[4][0] and pts[0][0]>pts[12][0] and pts[0][0]>pts[16][0] and pts[0][0]>pts[20][0]) and pts[5][0] > pts[4][0]:
        ch1=2
        #print("22222")
```

```
# condition for [c0][aemnst]
l = [[6,0],[6,6],[6,2]]
pl = [ch1, ch2]
if pl in l:
    if distance(pts[8],pts[16])<52:
        ch1 = 2
        #print("22222")


##print(pts[2][1]+15>pts[16][1])
# condition for [gh][bdfikruvw]
l = [[1,4],[1,5],[1,6],[1,3],[1,0]]
pl = [ch1, ch2]

if pl in l:
    if pts[6][1] > pts[8][1] and pts[14][1] < pts[16][1] and pts[18][1]<pts[20][1] and pts[0][0]<pts[8][0] and pts[0][0]<pts[12][0] and pts[0][0]<pts[16][0]
        ch1 = 3
        print("33333c")


#con for [gh][l]
l=[[4,6],[4,1],[4,5],[4,3],[4,7]]
pl=[ch1,ch2]
if pl in l:
    if pts[4][0]>pts[0][0]:
        ch1=3
        print("33333b")
```

```
# con for [gh][pqz]
l = [[5, 3],[5,0],[5,7], [5, 4], [5, 2],[5,1],[5,5]]
pl = [ch1, ch2]
if pl in l:
    if pts[2][1]+15<pts[16][1]:
        ch1 = 3
        print("33333a")

# con for [l][x]
l = [[6, 4], [6, 1], [6, 2]]
pl = [ch1, ch2]
if pl in l:
    if distance(pts[4],pts[11])>55:
        ch1 = 4
        #print("44444")

# con for [l][d]
l = [[1, 4], [1, 6],[1,1]]
pl = [ch1, ch2]
if pl in l:
    if (distance(pts[4], pts[11]) > 50) and (pts[6][1] > pts[8][1] and pts[10][1] < pts[12][1] and pts[14][1] < pts[16][1] and pts[18][1] <pts[20][1]):
        ch1 = 4
        #print("44444")
```

```
# con for [L][gh]
l = [[3, 6], [3, 4]]
pl = [ch1, ch2]
if pl in l:
    if (pts[4][0]<pts[0][0]):
        ch1 = 4
        #print("44444")

# con for [L][c0]
l = [[2, 2], [2, 5],[2,4]]
pl = [ch1, ch2]
if pl in l:
    if (pts[1][0] < pts[12][0]):
        ch1 = 4
        #print("44444")

# con for [L][c0]
l = [[2, 2], [2, 5], [2, 4]]
pl = [ch1, ch2]
if pl in l:
    if (pts[1][0] < pts[12][0]):
        ch1 = 4
        #print("44444")

# con for [gh][z]
l = [[3, 6],[3,5],[3,4]]
pl = [ch1, ch2]
if pl in l:
    if (pts[6][1] > pts[8][1] and pts[10][1] < pts[12][1] and pts[14][1] < pts[16][1] and pts[18][1] <pts[20][1]) and pts[4][1]>pts[10][1]:
        ch1 = 5
        print("55555b")
```

```
# con for [gh][pq]
l = [[3,2],[3,1],[3,6]]
pl = [ch1, ch2]
if pl in l:
    if pts[4][1]+17>pts[8][1] and pts[4][1]+17>pts[12][1] and pts[4][1]+17>pts[16][1] and pts[4][1]+17>pts[20][1]:
        ch1 = 5
        print("55555a")

# con for [L][pqz]
l = [[4,4],[4,5],[4,2],[7,5],[7,6],[7,0]]
pl = [ch1, ch2]
if pl in l:
    if pts[4][0]>pts[0][0]:
        ch1 = 5
        #print("55555")

# con for [pqz][aemnst]
l = [[0, 2],[0,6],[0,1],[0,5],[0,0],[0,7],[0,4],[0,3],[2,7]]
pl = [ch1, ch2]
if pl in l:
    if pts[0][0]<pts[8][0]  and  pts[0][0]<pts[12][0]  and pts[0][0]<pts[16][0]  and pts[0][0]<pts[20][0]:
        ch1 = 5
        #print("55555")
```

```
# con for [pqz][yj]
l = [[5, 7],[5,2],[5,6]]
pl = [ch1, ch2]
if pl in l:
    if pts[3][0]<pts[0][0]:
        ch1 = 7
        #print("77777")

# con for [l][yj]
l = [[4, 6],[4,2],[4,4],[4,1],[4,5],[4,7]]
pl = [ch1, ch2]
if pl in l:
    if pts[6][1] < pts[8][1]:
        ch1 = 7
        #print("77777")

# con for [x][yj]
l = [[6, 7],[0,7],[0,1],[0,0],[6,4],[6,6] ,[6,5],[6,1]]
pl = [ch1, ch2]
if pl in l:
    if pts[18][1] > pts[20][1]:
        ch1 = 7
        #print("77777")


# condition for [x][aemnst]
l = [[0,4],[0,2],[0,3],[0,1],[0,6]]
pl = [ch1, ch2]
if pl in l:
    if pts[5][0]>pts[16][0]:
        ch1 = 6
        #print("66666")
```

```
# condition for [yj][x]
l = [[7, 2]]
pl = [ch1, ch2]
if pl in l:
    if pts[18][1] < pts[20][1]:
        ch1 = 6
        #print("66666")


# condition for [c0][x]
l = [[2, 1],[2,2],[2,6],[2,7],[2,0]]
pl = [ch1, ch2]
if pl in l:
    if distance(pts[8],pts[16])>50:
        ch1 = 6
        #print("66666")

# con for [l][x]

l = [[4, 6],[4,2],[4,1],[4,4]]
pl = [ch1, ch2]
if pl in l:
    if distance(pts[4], pts[11]) < 60:
        ch1 = 6
        #print("66666")

#con for [x][d]
l = [[1,4],[1,6],[1,0],[1,2]]
pl = [ch1, ch2]
if pl in l:
    if pts[5][0] - pts[4][0] - 15 > 0:
        ch1 = 6
```

```
# con for [b][pqz]
l = [[5,0],[5,1],[5,4],[5,5],[5,6],[6,1],[7,6],[0,2],[7,1],[7,4],[6,6],[7,2],[5,0],[6,3],[6,4],[7,5],[7,2]]
pl = [ch1, ch2]
if pl in l:
    if (pts[6][1] > pts[8][1] and pts[10][1] > pts[12][1] and pts[14][1] > pts[16][1] and pts[18][1] > pts[20][1]):
        ch1 = 1
        print("111111")


# con for [f][pqz]
l = [[6, 1],[6,0],[0,3],[6,4],[2,2], [0,6],[6,2],[7, 6],[4,6],[4,1],[4,2], [0, 2], [7, 1], [7, 4], [6, 6], [7, 2], [7, 5], [7, 2]]
pl = [ch1, ch2]
if pl in l:
    if (pts[6][1] < pts[8][1] and pts[10][1] > pts[12][1] and pts[14][1] > pts[16][1] and
                    pts[18][1] > pts[20][1]):
        ch1 = 1
        print("111112")

l = [[6, 1], [6, 0],[4,2],[4,1],[4,6],[4,4]]
pl = [ch1, ch2]
if pl in l:
    if (pts[10][1] > pts[12][1] and pts[14][1] > pts[16][1] and
            pts[18][1] > pts[20][1]):
        ch1 = 1
        print("111112")
```

```
# con for [d][pqz]
fg=19
#print("_____________ch1=",ch1," ch2=",ch2)
l = [[5,0],[3,4],[3,0],[3,1],[3,5],[5,5],[5,4],[5,1],[7,6]]
pl = [ch1, ch2]
if pl in l:
    if ((pts[6][1] > pts[8][1] and pts[10][1] < pts[12][1] and pts[14][1] < pts[16][1] and
                    pts[18][1] < pts[20][1]) and (pts[2][0]<pts[0][0]) and pts[4][1]>pts[14][1]):
        ch1 = 1
        print("111113")

l = [ [4, 1], [4, 2],[4, 4]]
pl = [ch1, ch2]
if pl in l:
    if (distance(pts[4], pts[11]) < 50) and (pts[6][1] > pts[8][1] and pts[10][1] < pts[12][1] and pts[14][1] < pts[16][1] and pts[18][1] < pts[20][1]):
        ch1 = 1
        print("1111993")



l = [[3, 4], [3, 0], [3, 1], [3, 5],[3,6]]
pl = [ch1, ch2]
if pl in l:
    if ((pts[6][1] > pts[8][1] and pts[10][1] < pts[12][1] and pts[14][1] < pts[16][1] and
         pts[18][1] < pts[20][1]) and (pts[2][0] < pts[0][0]) and pts[14][1]<pts[4][1]):
        ch1 = 1
        print("1111mmm3")

l = [[6, 6],[6, 4], [6, 1],[6,2]]
pl = [ch1, ch2]
if pl in l:
    if pts[5][0]-pts[4][0]-15<0:
        ch1 = 1
        print("1111140")
```

```
# con for [i][pqz]
l = [[5,4],[5,5],[5,1],[0,3],[0,7],[5,0],[0,2],[6,2],[7, 5], [7, 1], [7, 6], [7, 7]]
pl = [ch1, ch2]
if pl in l:
    if ((pts[6][1] < pts[8][1] and pts[10][1] < pts[12][1] and pts[14][1] < pts[16][1] and
                pts[18][1] > pts[20][1])):
        ch1 = 1
        print("111114")

# con for [yj][bfdi]
l = [[1,5],[1,7],[1,1],[1,6],[1,3],[1,0]]
pl = [ch1, ch2]
if pl in l:
    if (pts[4][0]<pts[5][0]+15) and ((pts[6][1] < pts[8][1] and pts[10][1] < pts[12][1] and pts[14][1] < pts[16][1] and
                pts[18][1] > pts[20][1])):
        ch1 = 7
        print("111114lll;;p")

#con for [uvr]
l = [[5,5],[5,0],[5,4],[5,1],[4,6],[4,1],[7,6],[3,0],[3,5]]
pl = [ch1, ch2]
if pl in l:
    if ((pts[6][1] > pts[8][1] and pts[10][1] > pts[12][1] and pts[14][1] < pts[16][1] and
         pts[18][1] < pts[20][1])) and pts[4][1]>pts[14][1]:
        ch1 = 1
        print("111115")
```

```
# con for [w]
fg=13
l = [[3,5],[3,0],[3,6],[5,1],[4,1],[2,0],[5,0],[5,5]]
pl = [ch1, ch2]
if pl in l:
    if not(pts[0][0]+fg < pts[8][0] and pts[0][0]+fg < pts[12][0] and pts[0][0]+fg < pts[16][0]  and pts[0][0]+fg < pts[20][0]) and not(pts[0][0] > pts[8][0]
        ch1 = 1
        print("111116")

# con for [w]

l = [ [5, 0], [5, 5],[0,1]]
pl = [ch1, ch2]
if pl in l:
    if pts[6][1]>pts[8][1] and pts[10][1]>pts[12][1] and pts[14][1]>pts[16][1]:
        ch1 = 1
        print("1117")
```

```
if ch1 == 0:
    ch1='S'
    if pts[4][0] < pts[6][0] and pts[4][0] < pts[10][0] and pts[4][0] < pts[14][0] and pts[4][0] < pts[18][0]:
        ch1 = 'A'
    if pts[4][0] > pts[6][0] and pts[4][0] < pts[10][0] and pts[4][0] < pts[14][0] and pts[4][0] < pts[18][0] and pts[4][1] < pts[14][1] and pts[4][1] < pts[18][1]
        ch1 = 'T'
    if pts[4][1] > pts[8][1] and pts[4][1] > pts[12][1] and pts[4][1] > pts[16][1] and pts[4][1] > pts[20][1]:
        ch1 = 'E'
    if pts[4][0] > pts[6][0] and pts[4][0] > pts[10][0] and pts[4][0] > pts[14][0] and  pts[4][1] < pts[18][1]:
        ch1 = 'M'
    if pts[4][0] > pts[6][0] and pts[4][0] > pts[10][0]  and  pts[4][1] < pts[18][1] and pts[4][1] < pts[14][1]:
        ch1 = 'N'
```

```
if ch1 == 2:
    if distance(pts[12], pts[4]) > 42:
        ch1 = 'C'
    else:
        ch1 = 'O'

if ch1 == 3:
    if (distance(pts[8], pts[12])) > 72:
        ch1 = 'G'
    else:
        ch1 = 'H'

if ch1 == 7:
    if distance(pts[8], pts[4]) > 42:
        ch1 = 'Y'
    else:
        ch1 = 'J'

if ch1 == 4:
    ch1 = 'L'

if ch1 == 6:
    ch1 = 'X'

if ch1 == 5:
    if pts[4][0] > pts[12][0] and pts[4][0] > pts[16][0] and pts[4][0] > pts[20][0]:
        if pts[8][1] < pts[5][1]:
            ch1 = 'Z'
        else:
            ch1 = 'Q'
    else:
        ch1 = 'P'
```

```
if ch1 == 1:
    if (pts[6][1] > pts[8][1] and pts[10][1] > pts[12][1] and pts[14][1] > pts[16][1] and pts[18][1] >pts[20][1]):
        ch1 = 'B'
    if (pts[6][1] > pts[8][1] and pts[10][1] < pts[12][1] and pts[14][1] < pts[16][1] and pts[18][1] <pts[20][1]):
        ch1 = 'D'
    if (pts[6][1] < pts[8][1] and pts[10][1] > pts[12][1] and pts[14][1] > pts[16][1] and pts[18][1] > pts[20][1]):
        ch1 = 'F'
    if (pts[6][1] < pts[8][1] and pts[10][1] < pts[12][1] and pts[14][1] < pts[16][1] and pts[18][1] > pts[20][1]):
        ch1 = 'I'
    if (pts[6][1] > pts[8][1] and pts[10][1] > pts[12][1] and pts[14][1] > pts[16][1] and pts[18][1] < pts[20][1]):
        ch1 = 'W'
    if  (pts[6][1] > pts[8][1] and pts[10][1] > pts[12][1] and pts[14][1] < pts[16][1] and pts[18][1] < pts[20][1]) and pts[4][1]<pts[9][1]:
        ch1 = 'K'
    if ((distance(pts[8], pts[12]) - distance(pts[6], pts[10])) < 8) and (pts[6][1] > pts[8][1] and pts[10][1] > pts[12][1] and pts[14][1] < pts[16][1] and pts[18][
        ch1 = 'U'
    if ((distance(pts[8], pts[12]) - distance(pts[6], pts[10])) >= 8) and (pts[6][1] > pts[8][1] and pts[10][1] > pts[12][1] and pts[14][1] < pts[16][1] and pts[18]
        ch1 = 'V'

    if (pts[8][0] > pts[12][0]) and (pts[6][1] > pts[8][1] and pts[10][1] > pts[12][1] and pts[14][1] < pts[16][1] and pts[18][1] < pts[20][1]):
        ch1 = 'R'

if ch1== 1 or 'E' or 'S' or 'X' or 'Y' or 'B':
    if (pts[6][1] > pts[8][1] and pts[10][1] < pts[12][1] and pts[14][1] < pts[16][1] and pts[18][1] > pts[20][1]):
        ch1 = 'Space'

if ch1== 'E' or 'Y' or 'B':
    if (pts[4][0] < pts[5][0] ):
        ch1 = 'Next'

if ch1== 'Next' or 'B' or 'C' or 'H' or 'F':
    if (pts[0][0] > pts[8][0] and pts[0][0] > pts[12][0] and pts[0][0] > pts[16][0] and pts[0][0] > pts[20][0]) and pts[4][1]<pts[8][1] and pts[4][1]<pts[12][1] and
        ch1 = 'Backspace'
```

```
print("ch1=", ch1, " ch2=", ch2, " ch3=", ch3)
kok.append(ch1)

# # [0->aemnst][1->bfdiuvwkr][2->co][3->gh][4->l][5->pqz][6->x][7->yj]
if ch1 != 1:
    if (ch1,ch2) in dicttt:
        dicttt[(ch1,ch2)] += 1
    else:
        dicttt[(ch1,ch2)] = 1

frame = cv2.putText(frame, "Predicted " + str(ch1), (30, 80),
                    cv2.FONT_HERSHEY_SIMPLEX,
                    3, (0, 0, 255), 2, cv2.LINE_AA)
```

```
        cv2.imshow("frame", frame)
        interrupt = cv2.waitKey(1)
        if interrupt & 0xFF == 27:
            # esc key
            break


    except Exception:
        print("==", traceback.format_exc())



dicttt = {key: val for key, val in sorted(dicttt.items(), key = lambda ele: ele[1], reverse = True)}
print(dicttt)
print(set(kok))
capture.release()
cv2.destroyAllWindows()
```

References

[1] T. Yang, Y. Xu, and "A., Hidden Markov Model for Gesture Recognition", CMU-RI-TR-94 10, Robotics Institute, Carnegie Mellon Univ., Pittsburgh, PA, May 1994.

[2] Pujan Ziaie, Thomas M uller, Mary Ellen Foster, and Alois Knoll "A Na¨ive Bayes Munich, Dept. of Informatics VI, Robotics and Embedded Systems, Boltzmannstr. 3, DE-85748 Garching, Germany.

[3]https://docs.opencv.org/2.4/doc/tutorials/imgproc/gausian_median_blur_bilateral_filter/gausian_median_blur_bilateral_filter.html

[4] Mohammed Waleed Kalous, Machine recognition of Auslan signs using PowerGloves: Towards large-lexicon recognition of sign language.

[5]aeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural Networks-Part-2/

[6] http://www-i6.informatik.rwth-aachen.de/~dreuw/database.php

[7] Pigou L., Dieleman S., Kindermans PJ., Schrauwen B. (2015) Sign Language Recognition Using Convolutional Neural Networks. In: Agapito L., Bronstein M., Rother C. (eds) Computer Vision - ECCV 2014 Workshops. ECCV 2014. Lecture Notes in Computer Science, vol 8925. Springer, Cham

[8] Zaki, M.M., Shaheen, S.I.: Sign language recognition using a combination of new vision-based features. Pattern Recognition Letters 32(4), 572–577 (2011).

[9] N. Mukai, N. Harada and Y. Chang, "Japanese Fingerspelling Recognition Based on Classification Tree and Machine Learning," *2017 Nicograph International (NicoInt)*, Kyoto, Japan, 2017, pp. 19-24. doi:10.1109/NICOInt.2017.9

[10] Byeongkeun Kang, Subarna Tripathi, Truong Q. Nguyen" Real-time sign language fingerspelling recognition using convolutional neural networks from depth map" 2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR)

[11] Number System Recognition (https://github.com/chasinginfinity/number-sign-recognition)

[12] Mint (2021). 25 *Credit Card Fraud Statistics To Know in 2021 + 5 Steps for Reporting Fraud.* [online] MintLife Blog. Available at: https://mint.intuit.com/blog/planning/creditcard-fraud-statistics/.

[13] Asrar, S. (2022). *Debit, credit card frauds on rise, ATM scams down: NCRB.* [online] mint. Available at: https://www.livemint.com/industry/banking/debit-credit-card-fraudson-rise-atm-scams-down-ncrb-11661885877307.html [Accessed 26 Sep. 2022].

[14] citeseerx.ist.psu.edu. (n.d.). Download Limit Exceeded. [online] Available at: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.301.8231&rep=rep1&type=p df.

[15] Tuyls, Karl & Maes, Sam & Vanschoenwinkel, B.. (2022). Machine Learning Techniques for Fraud Detection