Siddhant Institute of Computer Application
PUNE-412109

PROJECT REPORT

On

"SIGN LANGUAGE TO TEXT & VOICE CONVERSION"

IN PARTIAL FULFILLMENT OF

MASTER OF COMPUTER APPLICATION

BY

KARTIK SATISH POOJARI

MCA I SEM II

2023-24

SUBMITTED TO

SAVITRIBAI PHULE PUNE UIVERSITY

ACKNOWLEDGEMENT

Apart from the efforts of myself, the success of the project depends largely on the encouragement and guidelines of many others. I take this opportunity to express my gratitude to the people who have been instrumental in the successful completion of this project.

I would like to express a deep sense of gratitude to **Director Prof. Nitin Shrirao** and our Head of the department **Prof. Reshma Mavkar** for their cordial support as they give the permission to use all required equipment and the necessary material to complete the project.

I would like to extend my sincerest gratitude to **Prof. Sujata Albhar** for her/his guidance and supervision as well as for providing necessary information regarding the project and also for the support in completing the project.

Finally I also extend my heartiest thanks to my parents, friends and well wishers for being with me and extending encouragement throughout the project.

CERTIFICATE

This is to certify that the Mini project report entitled, "**SIGN LANGUAGE TO TEXT & VOICE CONVERSION"** being submitted here with for the internal work of the degree of **MASTER OF COMPUTER APPLICATION (SEM-II)** to Savitribai Phule Pune University, Pune is the result of the original project work completed by **Kartik Satish Poojari** under the supervision and guidance of **Prof. Sujata Albhar** and to the best of my knowledge and belief, the work embodies in this Project has not formed earlier the basis for the award of any Degree of similar title or any other University or examining body.

Date:

Place: Pune

Prof. Sujata Albhar	**Prof. Reshma Mavkar**	**Prof. Nitin Shrirao**
Project Guide	**Head of Department**	**Director**

Internal Examiner	**External Examiner**
Name & Sign	Name & Sign

INDEX

INTRODUCTION

Computer vision tasks like sign translation and identification are essential for a variety of uses, such as autonomous vehicles, intelligent transportation systems, and accessibility for the blind [1]. In these exercises, the kind of sign that is present in an image must be identified, and the sign's meaning must be translated into a written or spoken language for ease of understanding. Communication is the act of conveying thoughts and messages using words, gestures, body language, and visual cues. To communicate with others, those who are deaf or dumb use a variety of hand signals. In nonverbal communication, gestures are used, and eyesight is used to interpret them. Sign language is a nonverbal method of interaction used by the dumb and deaf.
Sign language, like voice recognition, is a natural way of communication for persons who are deaf. Unfortunately, those who do not understand sign language generally underestimate or reject those who have this condition since they cannot effectively communicate with one another. This research is the first step in developing a skilled sign language translator able to receive messages in sign language and converting them to spoken language [2]. The project objective was to create a neural network that could recognise the letter of the alphabet being signed from an image of a signing hand. Many deaf and dumb people would find it much simpler to converse with others in social contexts if such a translator existed.

The goal of the project is to create a system that can convert Sign language into text and subsequently into voice. Communication between deaf and dumb people is made simpler because of the introduction of a simple, affordable method for reliably recognising sign language. By our technology, we were able to achieve 85% accuracy (both with and without a clear background and ideal lighting circumstances). We can even get 95% accuracy if the background is clear and the lighting is favourable.

OpenCV has been re-architected from C to modern, modular C++ compatible with STL and Boost. The library has been brought up to modern software development standards with distributed development on Git. Computer vision is an interdisciplinary scientific field that deals with how computers can be made to gain high-level understanding from digital images or videos. From the perspective of engineering, it seeks to automate tasks that the human visual system can do. OpenCV is a computer vision and machine learning software library that includes many common image analysis algorithms that will help us build custom, intelligent computer vision applications. In this application frequent image feed results in video tracking of our particular object of interest. Video tracking is the process of locating a moving object (or multiple objects) over time using a camera. It has a variety of uses human-computer interaction, security and augmented reality, traffic control, medical imaging, and video editing.

SCOPE OF PROJECT

- The Scope of this project is to create a system that can convert Sign Language into text.
- An easy-to-use and less expensive system for accurately identifying sign language is introduced, and communication between deaf and dumb people becomes easier.
- It uses an Image-based technique and can be run as an application in any basic system, resulting in a near-zero cost.

OPERATING ENVIRONMENT (HARDWARE AND SOFTWARE REQUIREMENTS):

- **Hardware Requirements:**
- Intel Core i5-3570.
- 8 GB Ram & above.

- **Software Requirements:**

- IDE Used : Visual Studio Code(Version 1.80.2)
- Language used: Python.
- Modules/Library used: Opencv, Mediapipe, TensorFlow, PyEnchant, Keras.

- **Functional Requirements**
  1. Collect a dataset of various sign languages.
  2. Extract the data that has been stored.
  3. Process data using Python libraries.

- **Non-Functional Requirements**

  Python workspace should be always updated with all the libraries that are required for running the process. Appropriate dataset should be given as an input.

PROPOSED SYSTEM

A proposed system of Sign Language Recognition and Translation using Deep Learning would involve leveraging the power of deep learning algorithms to understand and interpret sign language gestures. Here's a high-level overview of how such a system could be designed:

1. Data Collection: Collect a large dataset of sign language videos or images, capturing different gestures, signs, and expressions. This dataset should include diverse individuals performing sign language from various angles and perspectives to ensure robustness.
2. Data Preprocessing: Preprocess the collected dataset to extract relevant features or landmarks from the sign language videos or images. This may involve techniques like image resizing, normalization, background removal, and extracting key points or contours representing hand and body movements.
3. Model Architecture: Design a deep learning model suitable for sign language recognition and translation. Convolutional Neural Networks (CNNs) can be used to analyze image-based data, while Recurrent Neural Networks (RNNs) or Transformer models can handle temporal information for video-based data. Alternatively, a combination of these architectures can be used to capture both spatial and temporal features.
4. Training: Train the deep learning model using the preprocessed dataset. This involves feeding the input sign language videos or images along with their corresponding labels (e.g., corresponding words or phrases) into the model. The model learns to associate the visual features with the corresponding sign language gestures.
5. Sign Language Recognition: Once the model is trained, it can be used for real-time sign language recognition. Input video streams or images containing sign language gestures are passed through the trained model, which predicts the corresponding sign language gestures.
6. Translation: After recognizing the sign language gestures, the system can utilize a translation module to convert the recognized gestures into spoken or written language. This translation module can be based on rule-based approaches, machine translation techniques, or a combination of both.
7. User Interface: Develop a user-friendly interface for users to interact with the system. This can be a web or mobile application that allows users to input sign language gestures through a camera or upload videos/images. The application should display the recognized gestures and provide the translation output in real-time.
8. Evaluation and Iteration: Continuously evaluate and refine the system's performance based on user feedback and additional data collection. Fine-tuning the model and incorporating user suggestions can enhance the system's accuracy and usability.

OBJECTIVE OF SYSTEM

The objectives are as follows:

- To perform a neural network algorithm for Sign language recognition.
- To extract the Sign language data for future use.
- To implement the system using Deep learning.
- To show accuracy of the model.

USER REQUIREMENT

FEASIBILITY STUDY:

Feasibility study is useful to evaluate the cost and benefit of the system requested. This is nothing but a test, which is conducted on the proposed system, how the requirements are achieved.it is also used to understand the scope. The feasibility study tries to anticipate future scenario of system development. At this stage, the analyst estimates the urgency of the project and estimate the development cost. There are three major aspects in feasibility study:

1. Technical feasibility
2. Economic feasibility
3. Operational feasibility

1. Technical Study:

This study is carried out to check the technical feasibility that is the technical requirements of the system. Technical feasibility is concerned with the availability of hardware and system required for development of system, to see compatibility and maturity of the technology proposed to be used and to see the availability required the technical manpower to develop the system. After the study, we came to a conclusion that we proceed further with the tool and development environment chosen by us.

2. Economic Study:

Economic feasibility considers the cost benefits analysis of the proposed system. The benefits are always accepted to be overweighting the cost. Economic feasibility is helpful to find the system development cost hence and checks whether it is justifiable for that it concentrate on.

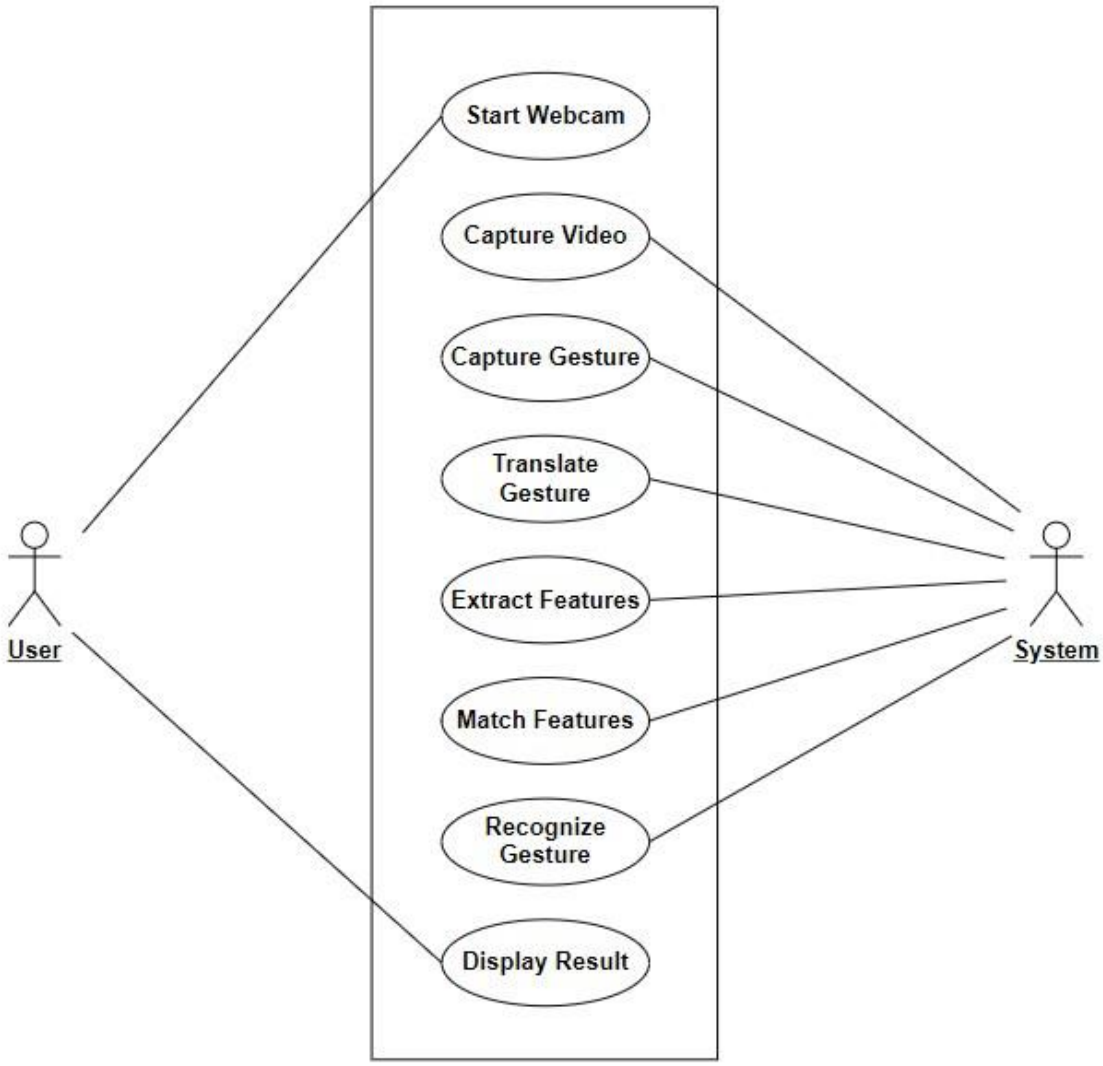
1. Investigation cost.
2. System and hardware cost.
3. Salaries and maintenance cost.
4. Supply cost. The proposed system requires minimum hardware and maintenance cost. Hence it is economic feasibility.

3. Operational study :-

It consider the acceptability of the system.it checks whether system will be used if it developed and implemented .it checks whether the users of the system will be able to handle as if the proposed system does not create any problems. The proposed system will be easy to handle as the overview is according to the requirement of the user. It reduces extra workload of the user proving to be beneficial. Hence the proposed system is operationally feasible. The level of acceptance by the users solely depends on the methods that are employed to educate the user about the system.

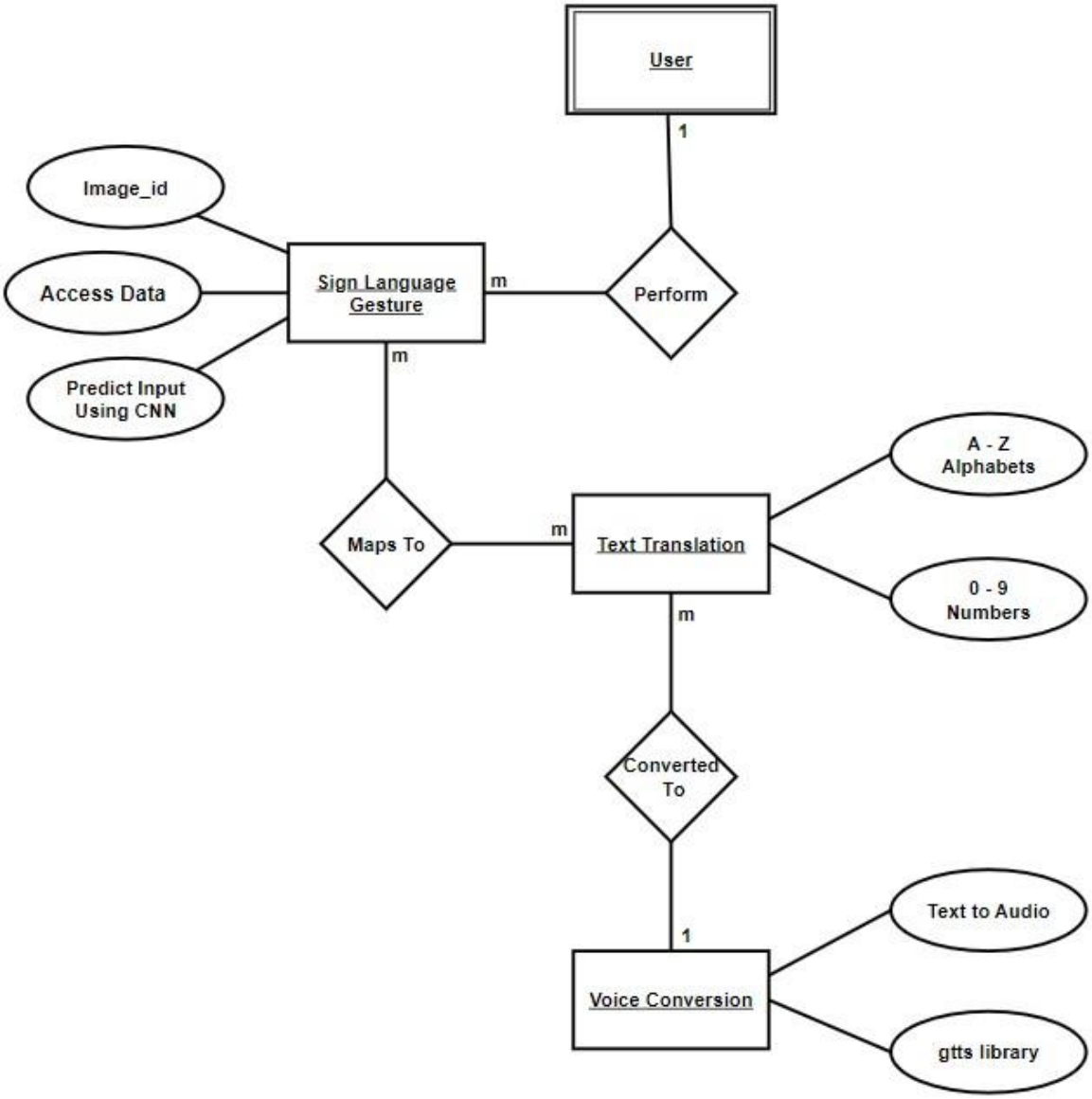
ANALYSIS & DESIGN

a) Use Case Diagram :



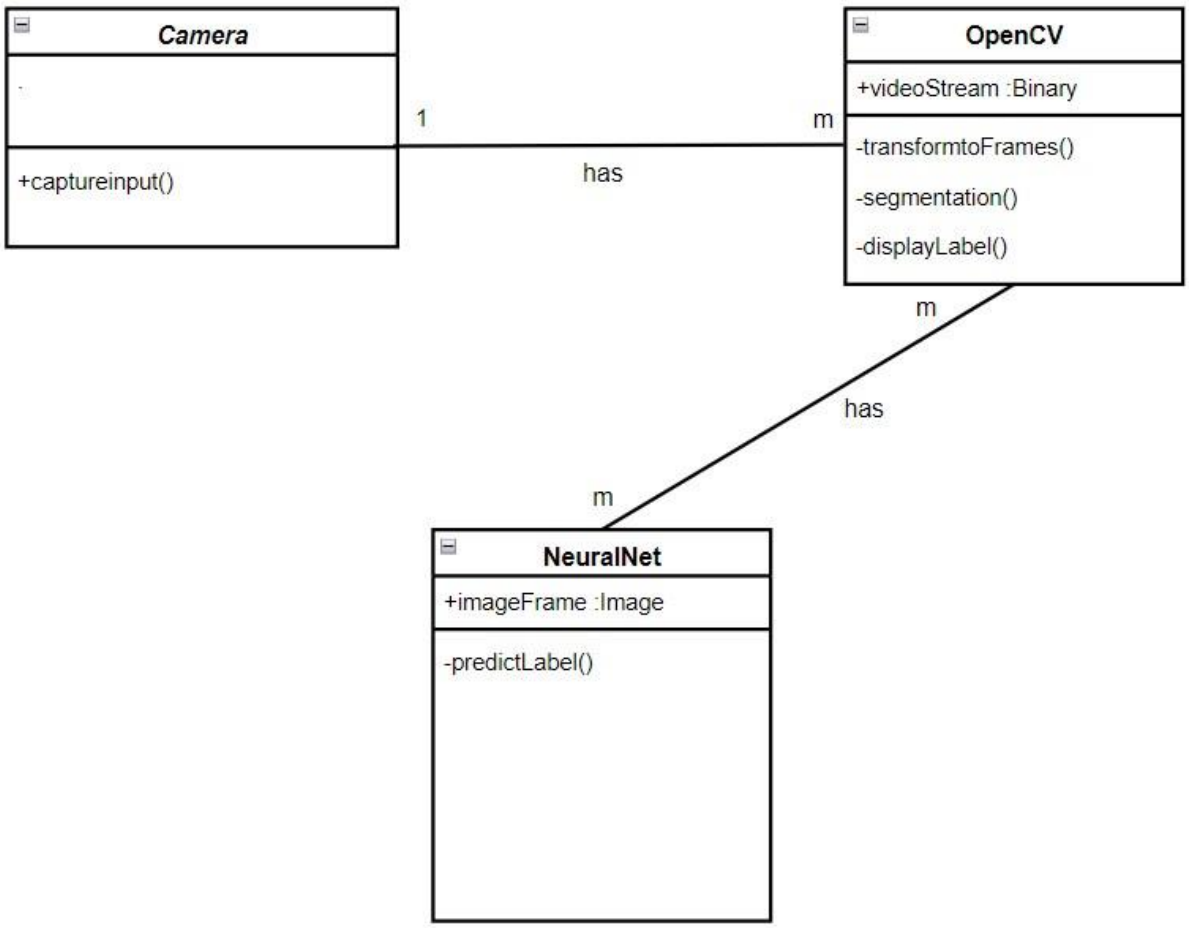
Use Case Diagram Of Sign Language To Text & Voice Conversion

b) ER Diagram (Entity Relationship Diagram) :



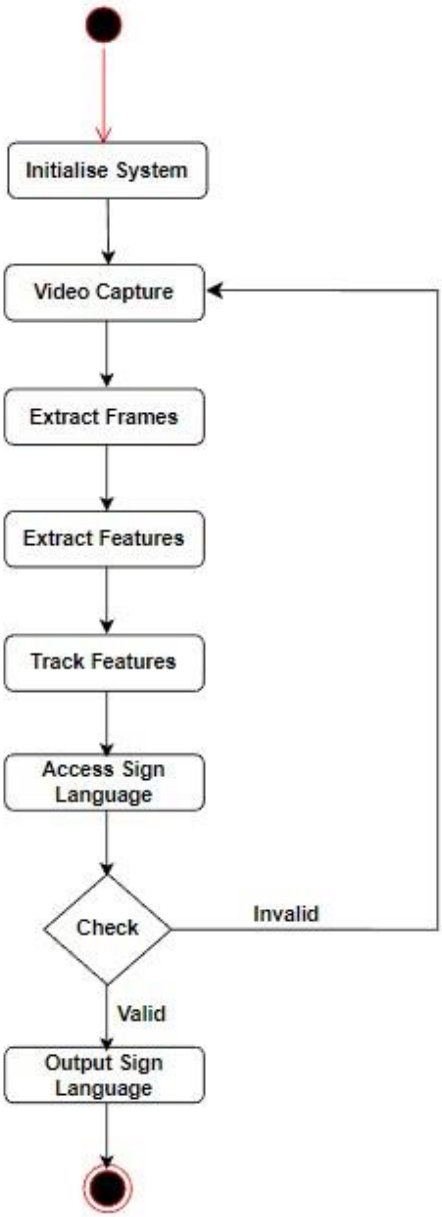
E-R Diagram Of Sign Language To Text & Voice Conversion

c) Class Diagram :



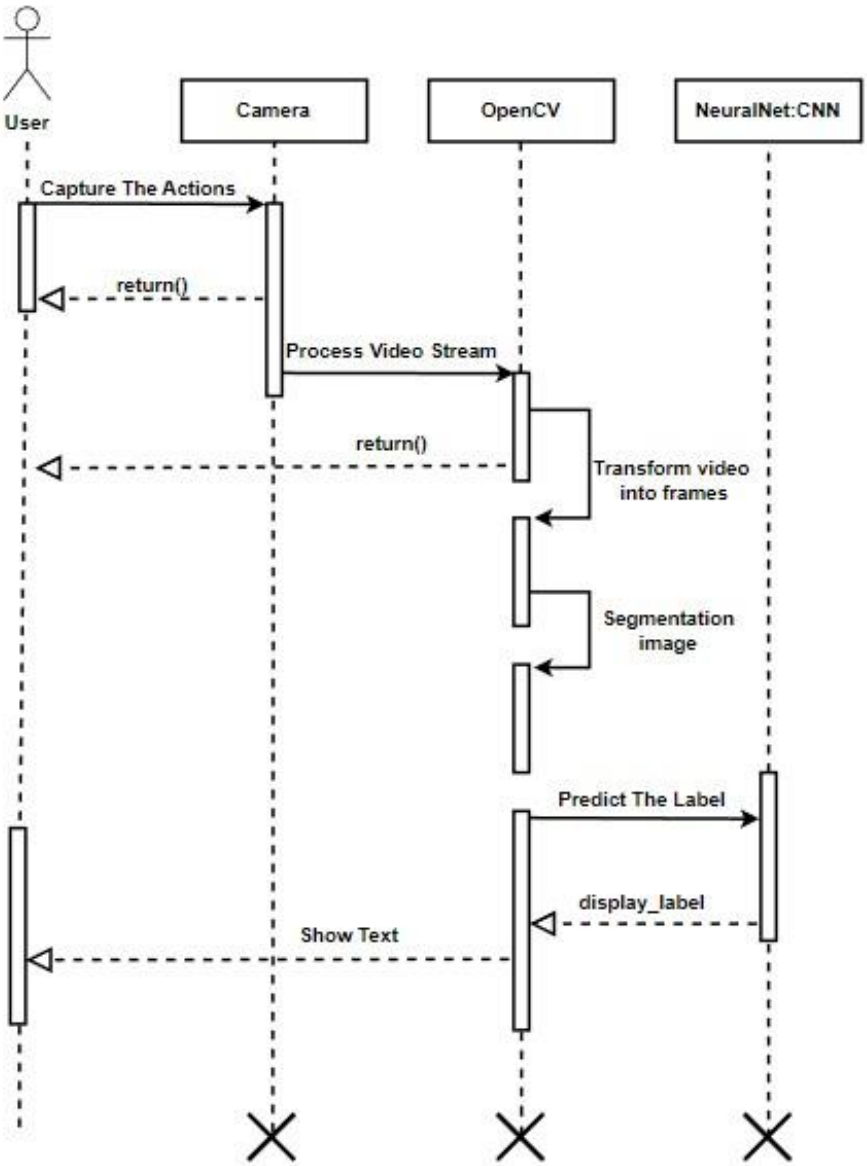
Class Diagram of Sign Language To Text & Voice Conversion

d) Activity Diagram :



Activity Diagram Of Sign Language To Text & Voice Conversion

c) Sequence Diagram :

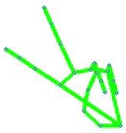


Sequence Diagram Of Sign Language To Text & Voice Conversion

User Interface Design (Screens etc.)

1. UI.PY OUTPUT :-

Sign Language Recognition and Translation

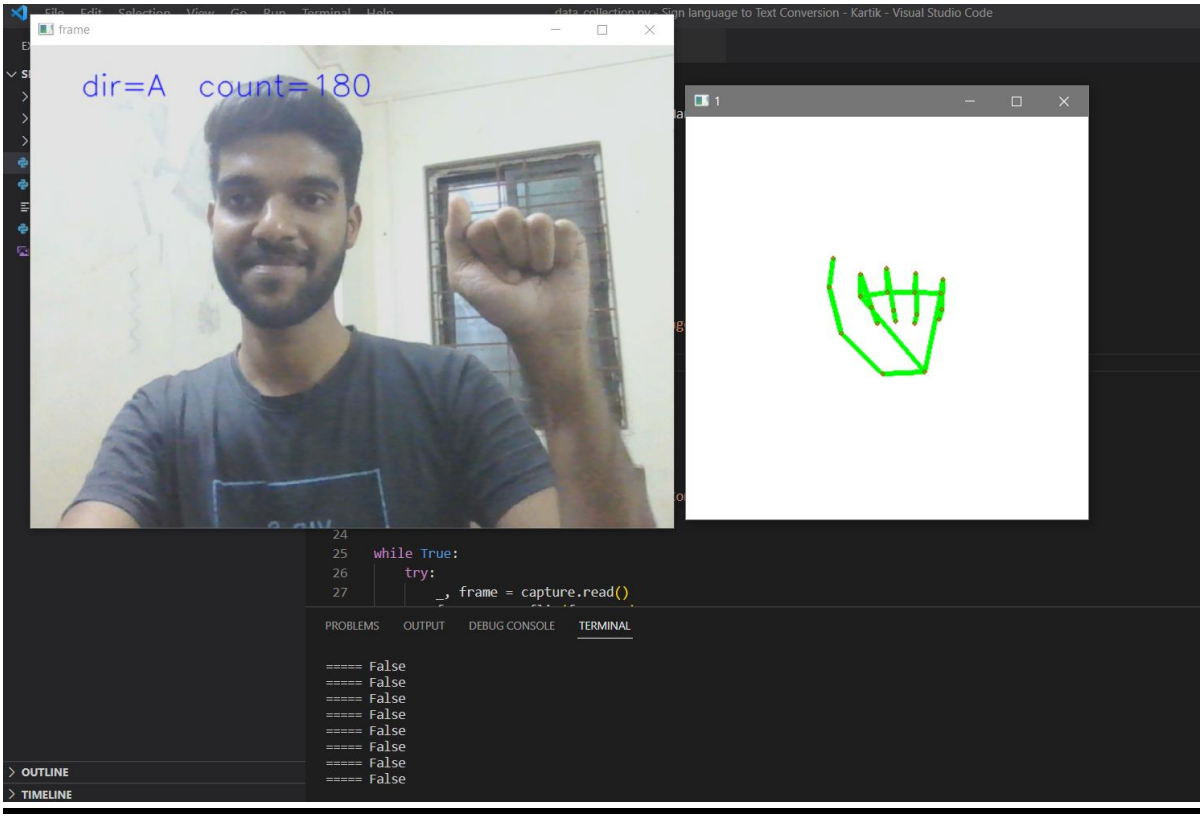


Character : Y

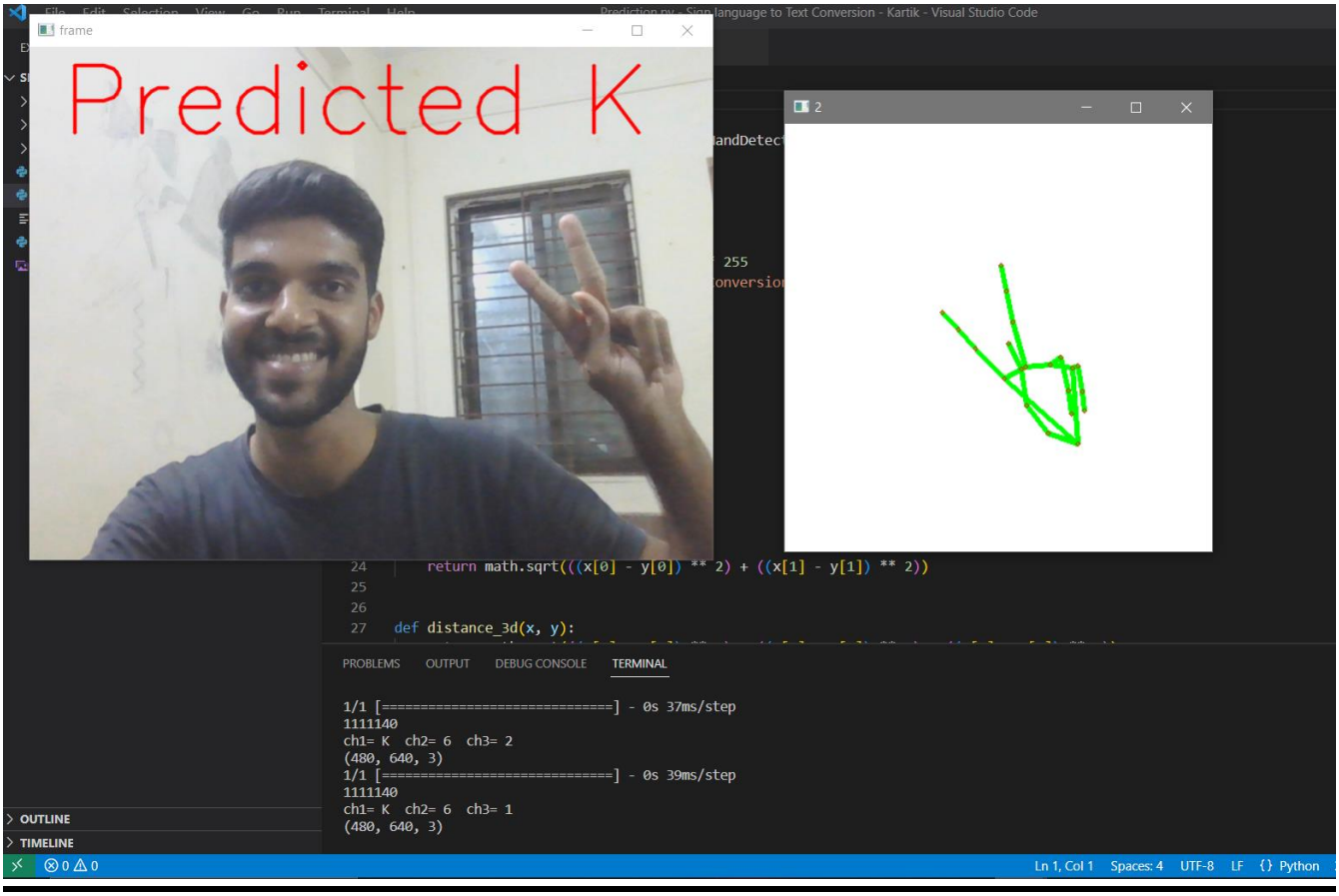
Sentence : SATYAWAHEESWARAN

Clear | Speak

2.DATA_COLLECTION.PY OUTPUT :-



3.PREDICTION.PY OUTPUT :-



4.Dataset :-

Figure below denotes The Dataset consists from Alphabet A to Z.

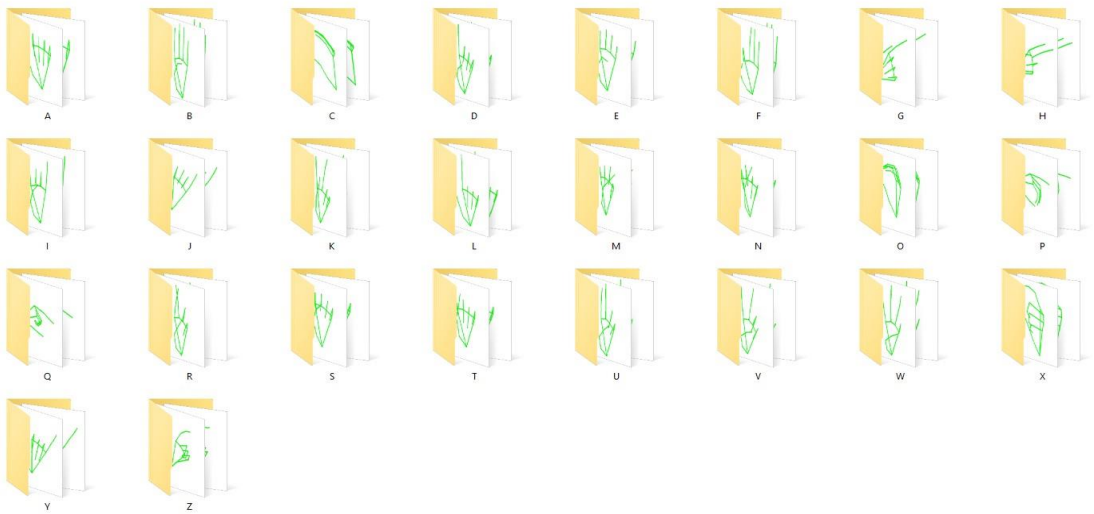


Figure 5.4: Dataset A to Z

Figure below display character A consists of 180 skeleton images.

Figure : Dataset of A

COMPONENTS USED IN THE PROJECT:

- IDE used in this project is VSCode.
- The components are hand-shape, palm orientation, the position of the hands, movement of the hands, and non-hand components such as the use of the face, facial expressions, or body posture.
- RNN Algorithm: A Recurrent Neural Network (RNN) algorithm is used to process the sequence of data points and predict the sign language gesture being performed. The RNN algorithm is trained using a dataset of labeled sign language gestures and can predict the gesture being performed with high accuracy.

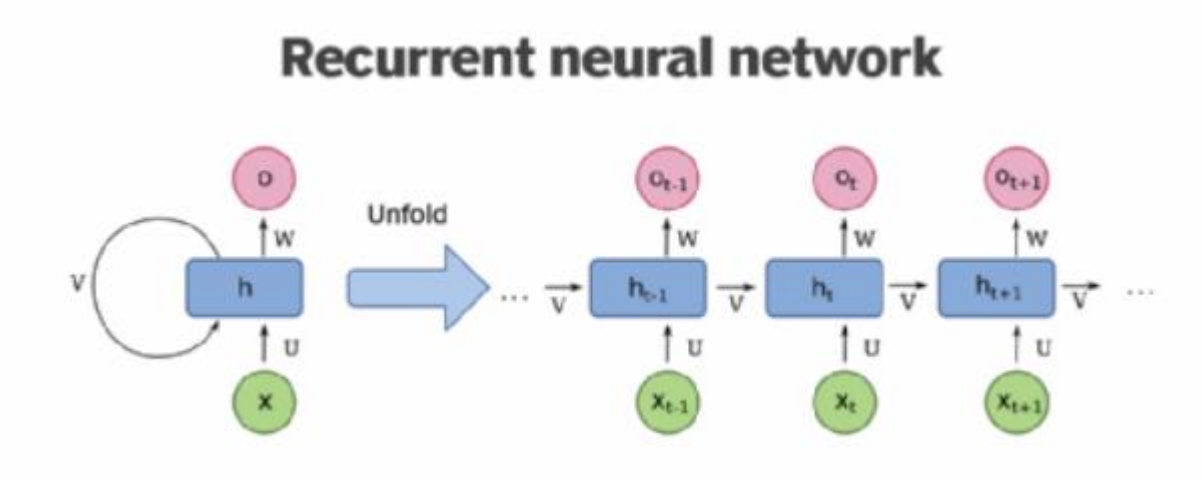


Figure : RNN Network

- A Convolutional Neural Network (CNN) is a kind of network architecture for deep learning algorithms and is specifically used for image recognition and tasks that involve the processing of pixel data. There are other types of neural networks in deep learning, but for identifying and recognizing objects, CNNs are the network architecture of choice.

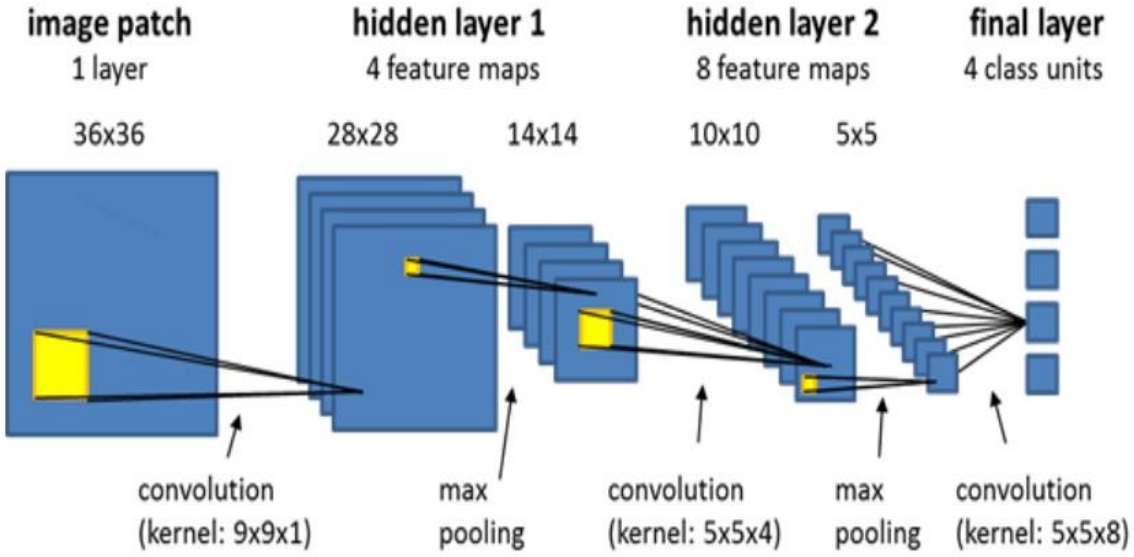


Figure : CNN Architecture

- **How is RNN different from CNN?**
  CNNs are commonly used in solving problems related to spatial data, such as images. RNNs are better suited to analyzing temporal, sequential data, such as text or videos.

- Using Mediapipe, we extract the hand from the frame and obtain the hand landmarks that are present in the image. We then draw and link those landmarks in a plain white image.
- We get these landmark points and draw them in a plain white background using the opencv library.
- By doing this we tackle the situation of background and lightning conditions because the mediapipe library will give us landmark points in any background and mostly in any lightning conditions.

- We have collected 180 skeleton images of Alphabets from A to Z.
- The preprocessed 180 images/alphabet will feed the keras CNN model.

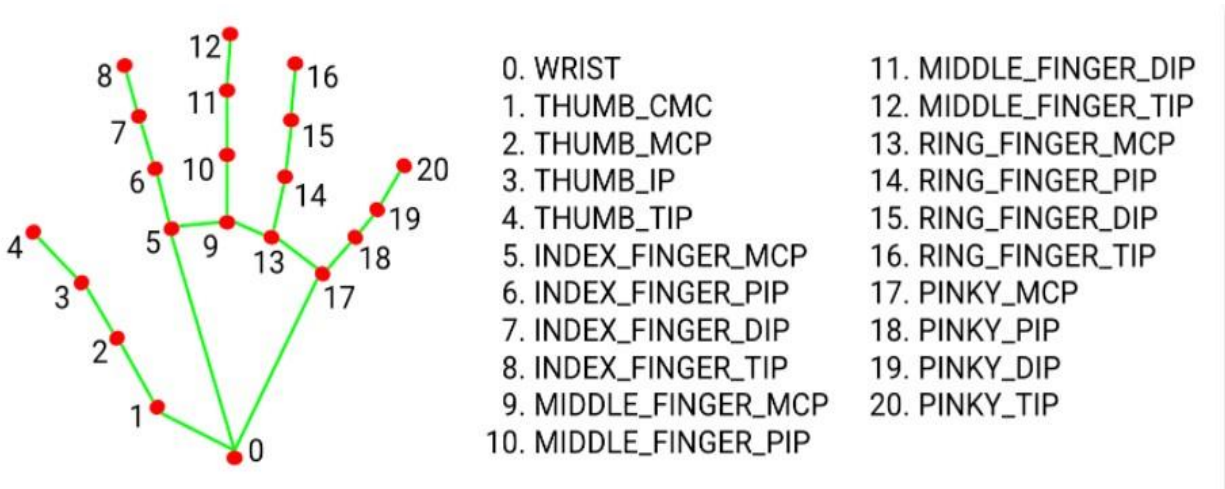


Figure : Mediapipe Landmark System

DRAWBACKS AND LIMITATIONS

- Sign language requires the use of hands to make gestures. This can be a problem for people who do not have full use of their hands.
- The system is very time consuming as it requires huge databases and hard-coding of combination to form these words.
- Speech synthesis consumes more processing power.
- Background and light conditions should be good to capture images.
- The type of speech needed for accurate results.
- GTTS may stop working for several reasons. The most common causes include software issues, incorrect settings, a poor internet connection (if your software needs internet to access voices), or compatibility issues. Furthermore, file corruption or a software bug may cause GTTS software to suddenly break.

FUTURE ENHANCEMENTS

- In future work, proposed system can be developed and implemented using Raspberry Pi.
- Image Processing part should be improved so that System would be able to communicate in both directions i.e.it should be capable of converting normal language to sign language and vice versa.
- We will try to recognize signs which include motion.
- Moreover, we will focus on converting the sequence of gestures into text i.e. word and sentences and then converting it into the speech which can be heard.
- This project, will help/motivate people to learn sign language in future.

CONCLUSION:

The project is focused on solving the problem of deaf and dumb people. This system will automate the hectic task of recognizing sign language, which is difficult to understand for a normal person, thus it reduces the efforts and increases time efficiency and accuracy. The system achieved an accuracy of 90% on the dataset, indicating its potential to facilitate communication between the deaf and the hearing. Using various concepts and libraries of image processing and fundamental properties of image we trying to develop this system. This project represented a visioned based system able to interpret hand gestures from the sign language and convert them into text & further convert to speech. The proposed system is tested in the real-time scenario, where it was possible to prove that obtained RNN models were able to recognize hand gestures. As future work is to keep improving the system and make experiments with complete language datasets.

REFERENCES AND BIBLOGRAPHY

[1] Gupta, A., Anpalagan, A., Guan, L., Khwaja, A.S.: Deep learning for object detection and scene perception in self-driving cars: Survey, challenges, and open issues. Array 10, 100057 (2021). https://doi.org/10.1016/j.array.2021.100057.

[2] Bragg, D., Koller, O., Bellard, M., Berke, L., Boudrealt, P., Braffort, A., Caselli, N., Huenerfauth, M., Kacorri, H., Verhoef, T., Vogler, C., Morris, M.: Sign Language Recognition, Generation, and Translation: An Interdisciplinary Perspective (2019).

[3] Kumar, A., Thankachan, K., Dominic, M.M.: Sign language recognition. In: 2016 3rd International Conference on Recent Advances in Information Technology (RAIT), pp. 422-428 (2016). https://doi.org/10.1109/RAIT.2016.7507939.

[4] Singh, J.P., Gupta, A., Ankita: Scientific Exploration of Hand Gesture Recognition to Text. In: 2020 International Conference on Electronics and Sustainable Communication Systems (ICESC), pp. 363-367 (2020). https://doi.org/10.1109/ICESC48915.2020.9155652.

[5] C.J., S., A., L.: Signet: A Deep Learning based Indian Sign Language Recognition System. In: 2019 International Conference on Communication and Signal Processing ,pp. 596-600 (2019). https://doi.org/10.1109/ICCSP.2019.8698006.

[6] Das, A., Gawde, S., Suratwala, K., Kalbande, D.: Sign Language Recognition Using Deep Learning on Custom Processed Static Gesture Images. In: 2018 International Conference on Smart City and Emerging Technology (ICSCET), pp. 1-6 (2018). https://doi.org/10.1109/ICSCET.2018.8537248.

[7] Fernandes, L., Dalvi, P., Junnarkar, A., Bansode, M.: Convolutional Neural Network based Bidirectional Sign Language Translation System. In: 2020 Third International Conference on Smart Systems and Inventive Technology (ICSSIT), Tirunelveli, India, pp. 769-775 (2020). https://doi.org/10.1109/ICSSIT48917.2020.9214272.

[8] Suharjito, Gunawan, H., Thiracitta, N., Nugroho, A.: Sign Language Recognition Using Modified Convolutional Neural Network Model. In: 2018 Indonesian Association for Pattern Recognition International Conference (INAPR), Jakarta, Indonesia, pp. 1-5 (2018). https://doi.org/10.1109/INAPR.2018.8627014.

[9] Patel, U., Ambekar, A.G.: Moment Based Sign Language Recognition for Indian Languages. In: 2017 International Conference on Computing, Communication, Control and Automation (ICCUBEA), Pune, India, pp. 1-6 (2017). https://doi.org/10.1109/ICCUBEA.2017.8463901.

[10] Bantupalli, K., Xie, Y.: American Sign Language Recognition using Deep Learning and Computer Vision. In: 2018 IEEE International Conference on Big Data (Big Data), Seattle, WA, USA, pp. 4896-4899 (2018). https://doi.org/10.1109/BigData.2018.8622141.

[11] Pahuja, D., Jain, S.: Recognition of Sign Language Symbols using Templates. In: 2020 8th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO), Noida, India, pp. 1157-1160 (2020). https://doi.org/10.1109/ICRITO48877.2020.9198001.

[12] Soodtoetong, N., Gedkhaw, E.: The Efficiency of Sign Language Recognition using 3D Convolutional Neural Networks. In: 2018 15th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON), Chiang Rai, Thailand, pp. 70-73 (2018). https://doi.org/10.1109/ECTICon.2018.8619984.

[13] More, S. S., Mange, M. A., Sankhe, M. S., & Sahu, S. S. (2021). Convolutional Neural Network based Brain Tumor Detection. In 2021 5th International Conference on Intelligent Computing and Control Systems (ICICCS).

[14] Heyman, J. (2021). Different pooling layers for CNN. Medium. Accessed March 16, 2023. https://jacobheyman702.medium.com/different-pooling-layers-for-cnn 4652a5103d6

ANNEXURE: SAMPLE PROGRAM CODE

UI.py

```
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
import enchant
ddd=enchant.Dict("en-US")
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('sign_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 Recognition and Translation")
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 Recognition and Translation", 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.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("D:\Sign language to Text Conversion - Kartik\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.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.pts[18][1] > self.pts[20][1]):
            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[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]) and (self.pts[4][1]
< self.pts[6][1] and self.pts[4][1] < self.pts[10][1] and self.pts[4][1] < self.pts[14][1] and self.pts[4][1]
< self.pts[18][1]):
            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()
```

data_collection.py

```
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("D:\Sign language to Text Conversion - Kartik\Data\A"))
c_dir = 'A'

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

white=np.ones((400,400),np.uint8)*255
cv2.imwrite("D:\Sign language to Text Conversion - Kartik\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("D:\Sign language to Text Conversion - Kartik\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("E:\Major_Project\images" + (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("D:\Sign language to Text Conversion - Kartik\Data\A" + (c_dir) + "\\" +
str(count) + ".jpg",
                            skeleton1)

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



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

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

Prediction.py

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

model = load_model('sign_model.h5')
white = np.ones((400, 400), np.uint8) * 255
cv2.imwrite("D:\Sign language to Text Conversion - Kartik\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("D:\Sign language to Text Conversion - Kartik\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],[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 (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] and pts[0][0]<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 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] and pts[0][0] > pts[12][0] and pts[0][0] >
pts[16][0]  and pts[0][0] > pts[20][0]) and distance(pts[4], 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 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][1] < pts[20][1]):
                    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][1] < pts[20][1]) and (pts[4][1]
>pts[9][1]):
                    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 pts[4][1]<pts[16][1] and
pts[4][1]<pts[20][1]:
                    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:
```

```
                dictttt[(ch1,ch2)] += 1
            else:
                dictttt[(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())



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

THANK YOU