TABLE OF CONTENTS

ABSTRACT

This presents the development of a backend system with graphical user interfaces (GUIs) for a comprehensive sign language translation application. The system utilizes machine learning, computer vision, and natural language processing techniques implemented in Python programming language to enable text to sign language conversion and sign language to text detection.

The backend system consists of several GUI components that provide an intuitive and user-friendly interface for users to interact with the application. Users can input text messages through the GUIs, which are then processed using natural language processing techniques. The system analyses and interprets the input text, generating corresponding sign language animations/ videos.

To achieve accurate sign language to text detection, the backend system leverages machine learning models trained on diverse sign language datasets generated using Hand-Tracking support of computer vision. These models initially predict the Sign Language symbol and on basis of it suggest words or sentences.
Additionally, the system incorporates computer vision algorithms to detect and interpret sign language gestures captured through the smartphone's camera. The GUIs display the recognized gestures as text, enabling sign language to text detection. This functionality facilitates bi-directional communication between sign language users and non-signing individuals.

The implementation of this backend system with GUIs demonstrates the power of combining machine learning, computer vision, and natural language processing in sign language translation. By providing reliable text to sign language conversion and sign language to text detection, the application aims to bridge communication barriers and promote inclusivity for individuals who rely on sign language as their primary means of expression.

1. INTRODUCTION

1.1 Problem Statement: Communication with deaf people continues to be a significant challenge for individuals without hearing impairments. While various online methods exist to facilitate communication, such as messaging, texting, sign language hand gestures, and hearing impairment machines, each approach has its limitations. For example, messaging requires knowledge of the language being used, which can be a barrier for deaf individuals. Sign language, on the other hand, requires extensive practice to master hand gestures, and hearing impairment machines can be expensive and not accessible to everyone.

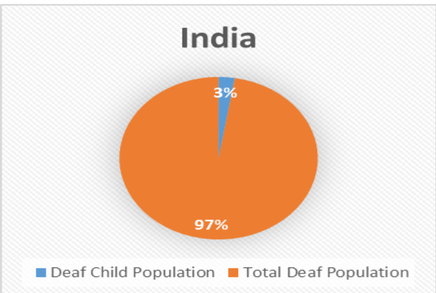
Moreover, the representation of hearing-impaired individuals in educational institutions is disproportionately low. According to a report by Hindustan Times, only 5% of hearing-impaired children in our country attend school. This lack of inclusion poses further challenges for effective communication and participation in various fields.

To address these issues, I conducted research and discovered that existing solutions are not comprehensive or satisfactory. This realization led me to develop SignCompanion, a backend program designed to facilitate communication with deaf individuals.

By acting as a facilitator, SignCompanion aims to bridge the communication gap. It leverages technology to provide a user-friendly and efficient platform for communication. The program considers the limitations of existing methods and strives to overcome them.

In conclusion, SignCompanion is a backend program that addresses the communication challenges faced by deaf individuals. Through innovative and accessible features, it aims to enhance communication and promote inclusivity for the hearing impaired.

- In India Total Hearing Impaired/Deaf Population -63 M / 1.39 B
- In India Total Child Hearing Impaired/child Deaf Population -1.7 M / 63 M



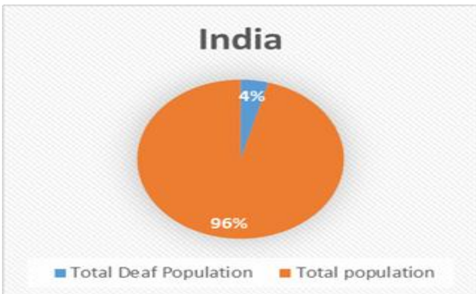


Fig – Hearing impaired population in India

1.2 Approach: SignComapnion – the backend program has two options available and the approach differ according to option

- Text or Speech-to-Sign Language Conversion
- Sign Language to text Detection and suggestion

2. Text or Speech-to-Sign Language Conversion:

This option utilizes object-oriented programming in Python and GUI libraries to create a user-friendly interface. Users have the option to input text or even speak into a microphone to convert their words into sign language hand gestures.

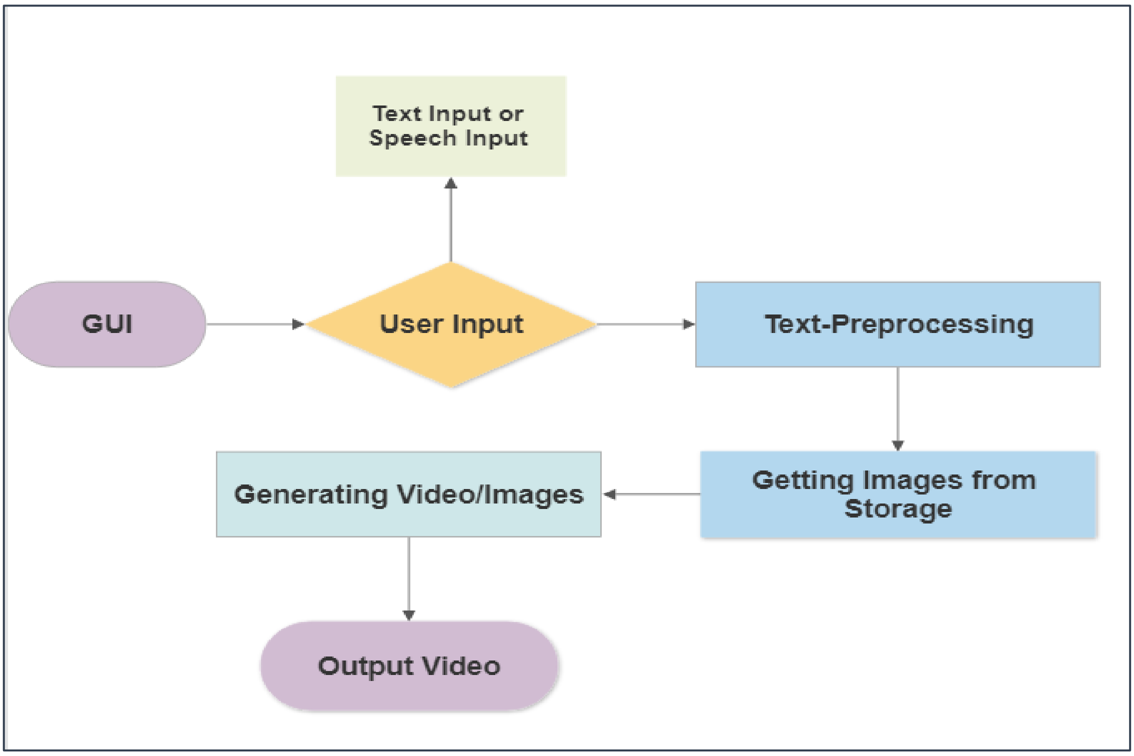
The entered text or speech is then subjected to a pre-processing step using Natural Language Processing (NLP) techniques. This helps to normalize and standardize the text, ensuring it is in a suitable format for further processing.

Next, the system iterates over each letter in the processed text and retrieves corresponding images or visuals from a library or folder. These images represent the hand gestures for each letter of the alphabet or other characters.

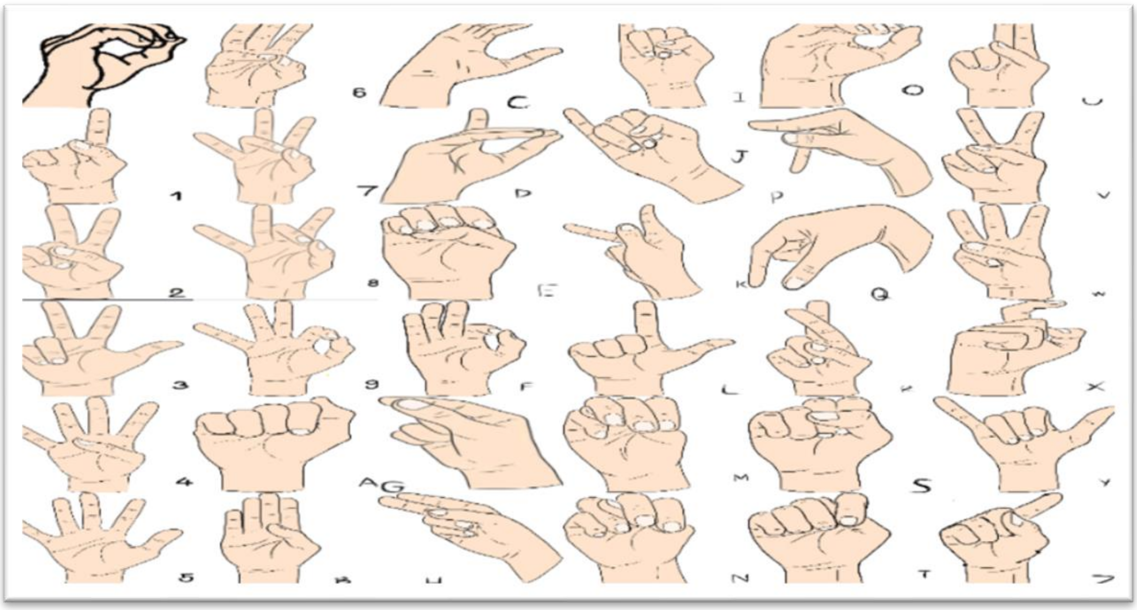
To create a complete video representation of the entered text, these individual letter images are processed using OpenCV, a computer vision library. OpenCV allows us to manipulate and combine the images to generate a video. The video consists of frames, with each frame representing a different letter from the input text. As the frames are played in sequence, it gives the illusion of a continuous video showing the sign language representation of the entire text.

In summary, this approach uses Python programming and GUI libraries to enable users to input text or speech and convert it into sign language hand gestures. Through pre-processing, image retrieval, and OpenCV video generation, the system creates a visual representation of the input text, allowing for improved communication with deaf individuals.

2.1 Flow Diagram:



2.2 Sign Language Images Data: Images for sign languages are easily available on Google.



2.3 Program :

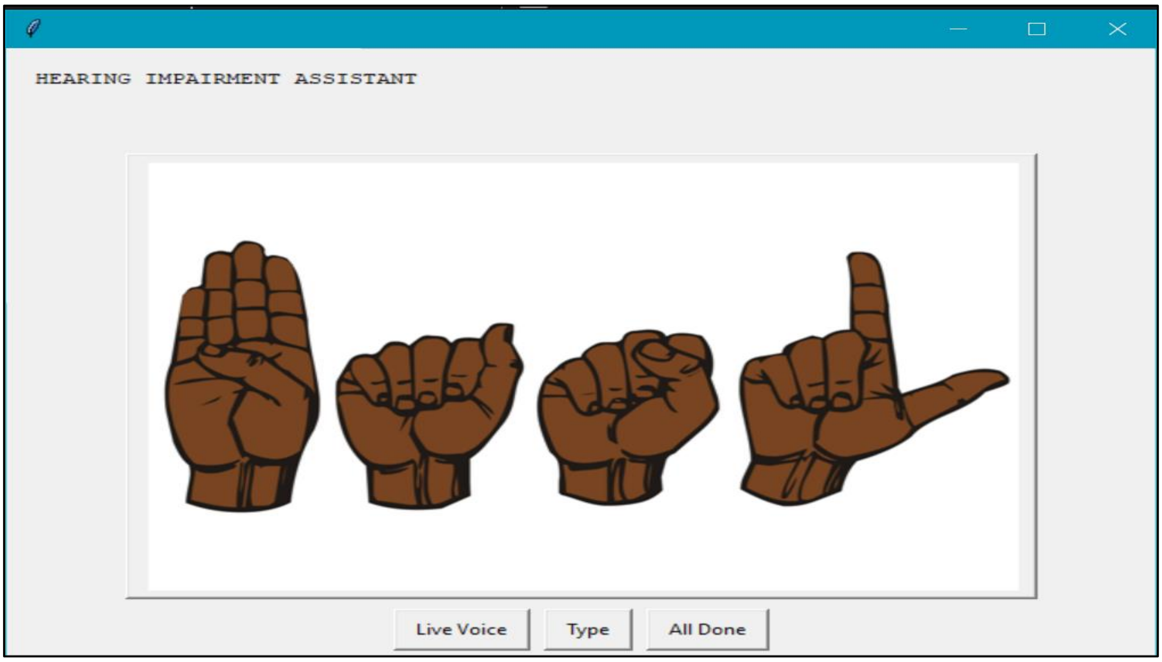
```python
#libraries required

import speech_recognition as sr
import os
import numpy as np
import cv2
from easygui import *
import string

#class to combine various function inside one class

class Speech_to_asl:

    def __init__(self) -> None:
        self.images_path = 'D:/my python/python code/speech/letter'
        self.image_files = os.listdir(self.images_path)

    def text_preprocessing(self,sequence):
        sequence=sequence.lower()
        for c in string.punctuation:
            sequence =sequence.replace(c,"")
        return sequence

    def image_func(self,text):

        #initializing the video writer to create video
        video_writer = cv2.VideoWriter('output.mp4',cv2.VideoWriter_fourcc(*'mp4v'),0.4,(640,480))
        images_list=[]
        for letter in text:
            for image_file in self.image_files:
                if image_file[0] == letter:
                    image=cv2.imread(f'D:/my python/python code/speech/letter/{image_file}') ## 0 for cv2.IMREAD_GRAYSCALE
                    img=cv2.resize(image,(640,480))
```

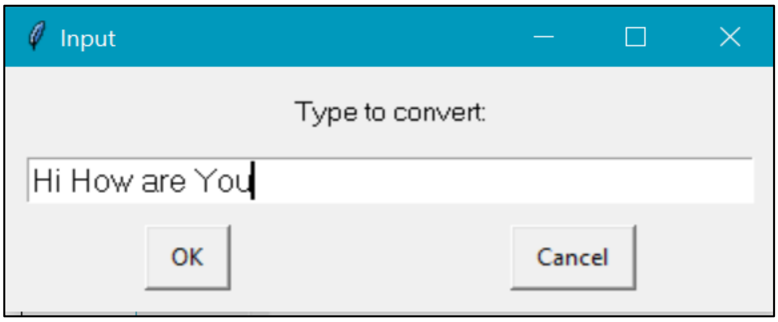
```python
        for i in range(1):
            blank_frame = np.zeros((480, 640, 3), np.uint8)
            video_writer.write(blank_frame)

        video_writer.release()
        # Set the frame rate of the video
        frame_rate = 10

    # Create a named window for displaying the video
        cv2.namedWindow('Video', cv2.WINDOW_NORMAL)
        # Set the display window's size
        cv2.resizeWindow('Video', 500, 500)
        # Use a higher quality interpolation method
        interpolation = cv2.INTER_LINEAR

        for img in images_list:
            # Convert the image to a format that OpenCV can display
            img = cv2.resize(img, (640, 480), interpolation=interpolation)
            # Display the image in the named window
            cv2.imshow('Video', img)

            # Wait for the specified time (in milliseconds) between each frame
            delay = int(10000 / frame_rate)
            cv2.waitKey(delay)

        # Release the named window
        cv2.destroyWindow('Video')

    def live_voice_func(self):

            r = sr.Recognizer() #initalizing the microphone
            with sr.Microphone() as source:
                r.adjust_for_ambient_noise(source)
                while True:
```

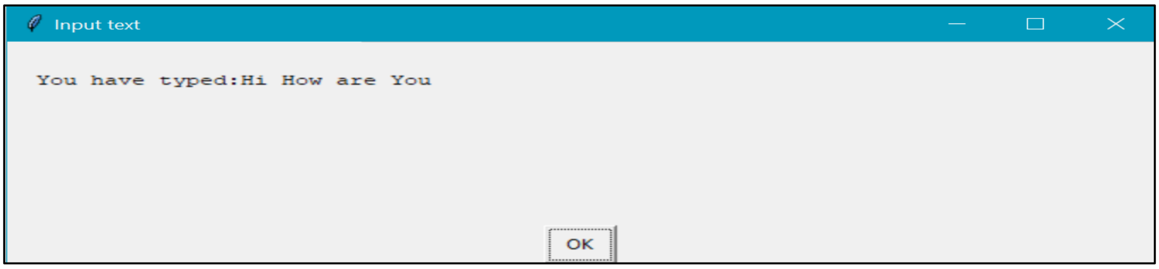
```python
                    # recognize speech using Sphinx
                    try:
                        msgbox("Speak something to convert"+'\n'+"I am Listening.....",title="speech recording")
                        audio = r.listen(source)
                        text=r.recognize_google(audio) #recognizing the voice
                        msgbox("You have said: "+text,title='Input text')
                        print('You Said: ' + text.lower())
                        if(text=='goodbye' or text=='good bye' or text=='bye' or text =='exit'):
                            msgbox("oops!Time To say good bye",title="Exit")
                            print("oops!Time To say good bye")
                            break
                        else:
                            text=self.text_preprocessing(text)
                            self.image_func(text)
                            continue
                    except:
                        print("oops error")
                        break

    def text_func(self):
        while True:
            #sent=str(input('--- type something ---'))
            sent=enterbox("Type to convert:",title="Input")
            msgbox("You have typed:"+str(sent),title='Input text')
            sent=self.text_preprocessing(str(sent))
            if(sent=='goodbye' or sent=='good bye' or sent=='bye' or sent=='exit'):
                msgbox("oops!Time To say good bye",title="Exit")
                print("oops!Time To say good bye")
                break
            else:
                self.image_func(sent)
                continue
```

```python
    def main(self):
        print("Starting Application...")
        while 1:
            image   ="D:\my python\python code\speech\signlang.png"
            msg="HEARING IMPAIRMENT ASSISTANT"
            choices = ["Live Voice","Type","All Done"]
            reply   = buttonbox(msg,image=image,choices=choices)
            print(reply)
            #reply=str(input(f'choose any of this {choices}'))
            if reply ==choices[0]:
                    self.live_voice_func()
            elif reply == choices[1]:
                    self.text_func()
            else:
                    msgbox("oops!Time To say good bye"+'\n'+"See You again",title="Exit",)
                    print("see you again !!! ")
                    break

        print("Closing Application...")


#accessing main() to run other functions
(Speech_to_asl()).main()
```

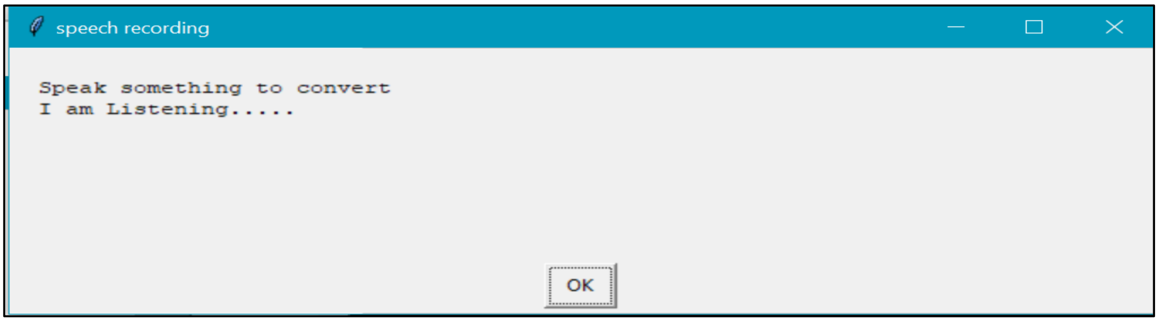
2.4 Output:

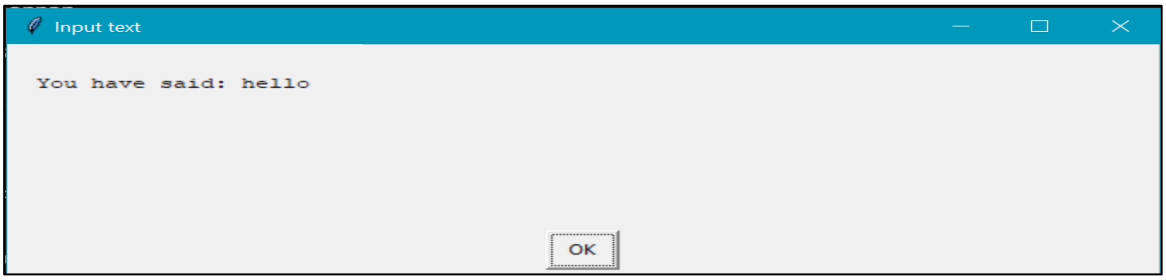






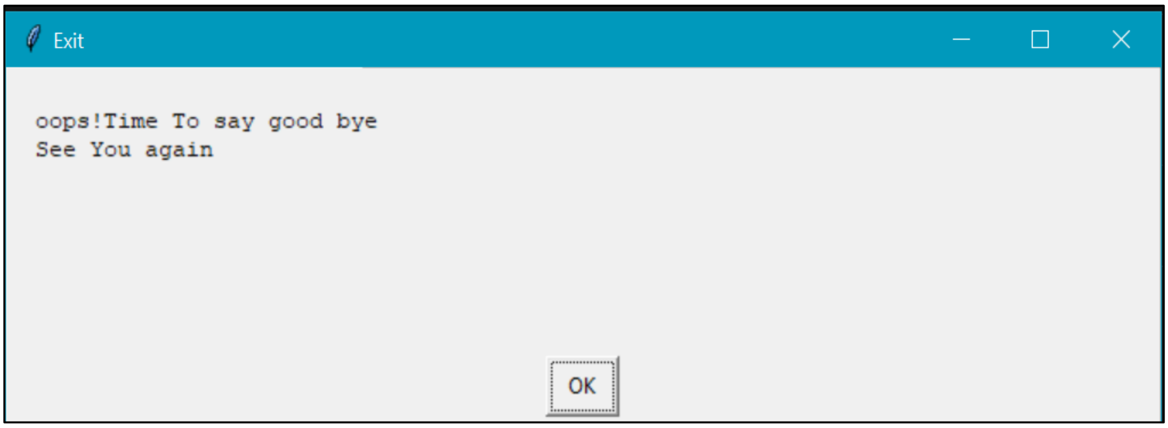
So, this will generate a video of all the letters in the entered text. Here is the glimpse of the video frame that will play after delay so that user can easily see them.





So, On press of OK this will generate a video of all the letters in the entered text. Here is the glimpse of the video frame that will play after delay so that user can easily see them.

And to exit from the GUI either press “All Done” or speak/type “Bye” “Good Bye” “Exit”.



2.5 Future Improvement:

In addition to selecting images from a storage library, an advanced approach involves training a model that can generate image sequences based on a given text input. This approach utilizes deep learning techniques, specifically Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) networks.

CNNs are used to extract relevant features from image sequences that correspond to the text input. These features capture important visual information related to each letter in the sequence. On the other hand, LSTM networks serve as memory recorders, associating text features with each image in the text sequence. This helps the model learn the relationship between the text and the corresponding images.

To train such a model, a large and diverse dataset is required. The dataset consists of pairs of text sequences and the corresponding arrays of images representing each letter in the sequence. Training a model on this data demands significant computational resources, including powerful GPUs, as the processing requirements for deep learning tasks can be intensive.

As an example, the training dataset would be structured as follows:
Train Dataset / Test Data: {"text sequence": "list of arrays of images for each letter in the sequence"}

By training the model on this dataset, it can learn to generate image sequences that accurately correspond to a given text input. This advanced approach leverages the power of deep learning and allows for more dynamic and flexible generation of sign language hand gestures based on text input.

3. Sign Language to text Detection and suggestion:

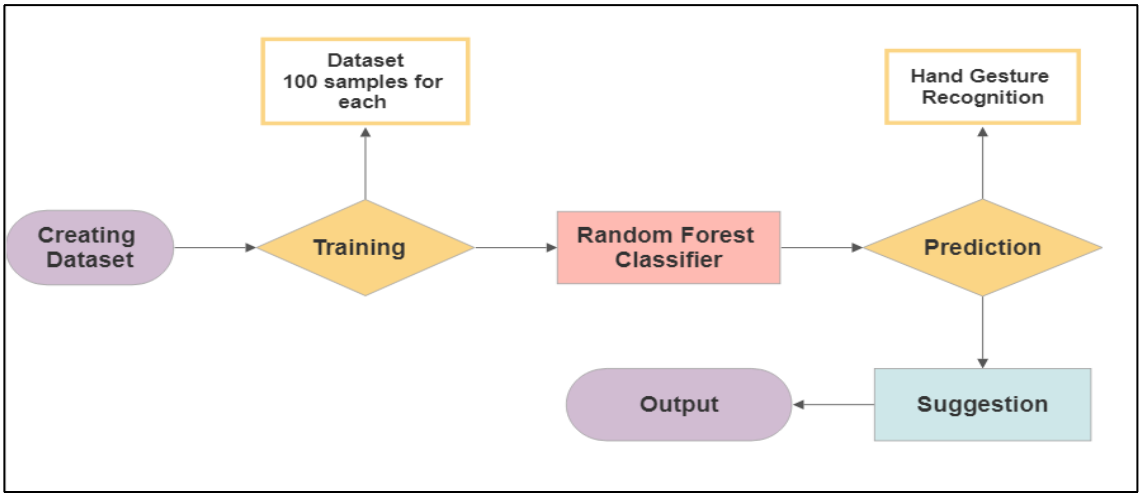
To facilitate the recognition and interpretation of sign language hand gestures, I incorporated machine learning and computer vision concepts into the application.

Firstly, I trained a machine learning classification model using a combination of machine learning algorithms and computer vision techniques. This model is capable of recognizing and classifying different hand gestures corresponding to various letters of sign languages. The Hand Tracking function from the OpenCV library was utilized to accurately track and capture hand movements.

Once the model predicts the letter based on the hand gesture, the application suggests the most associated word or sentence starting with that letter. To achieve this, I created separate pickle files that store a list of common words and sentences related to sign language. These files serve as references for generating appropriate suggestions based on the recognized letter.

By incorporating machine learning and computer vision, the application is able to accurately classify hand gestures and provide relevant word or sentence suggestions, enhancing the overall communication experience for users of sign language.

3.1 Flow Diagram:



3.2 Data Set:

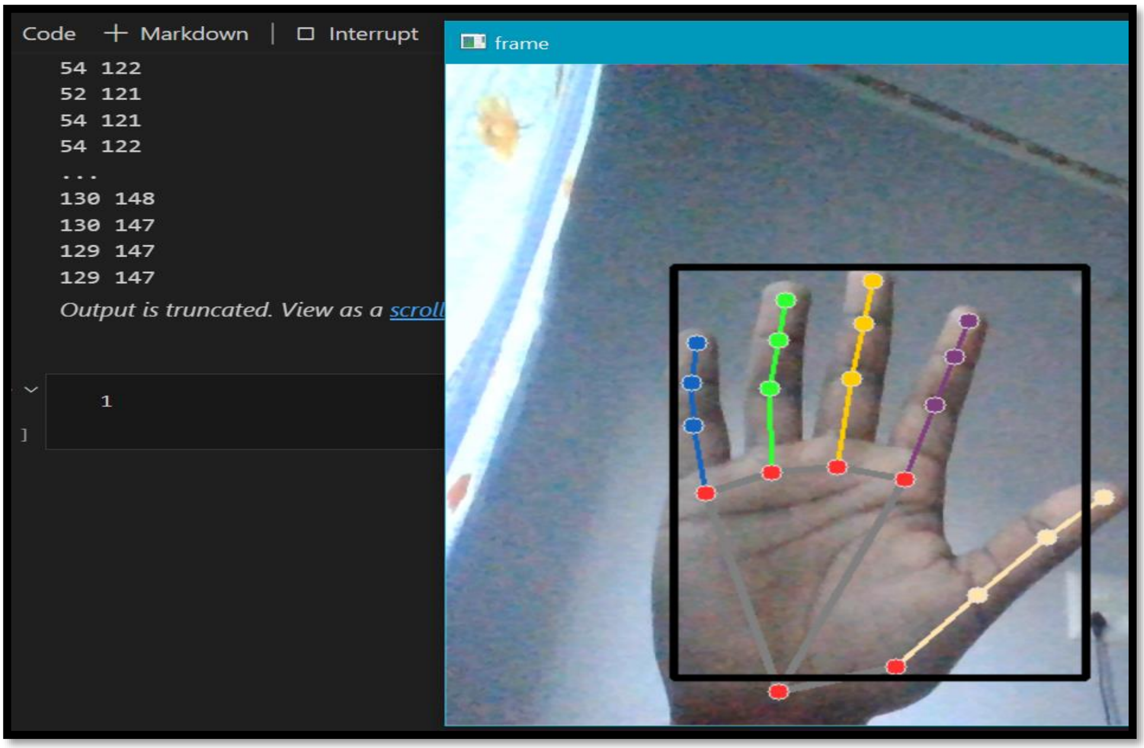
To create a dataset for sign language symbols, including alphabets and digits, I utilized the OpenCV library. I captured a series of hand gestures by making the respective sign for each symbol and extracted individual frames using OpenCV methods. In total, I collected 100 sample images per symbol (36), which served as the foundation for training the machine learning model.

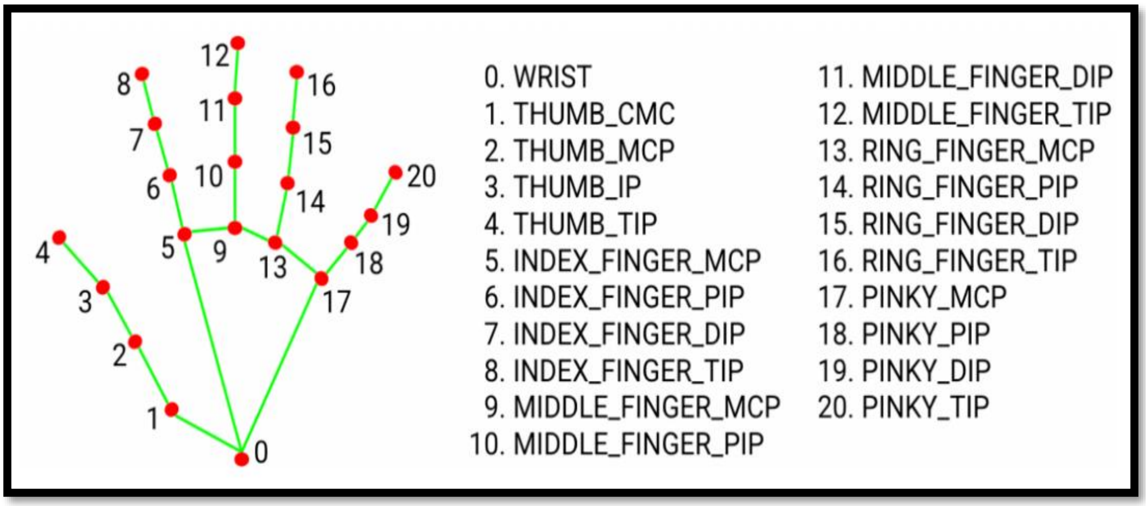
```python
create.py > ...
#importing the required libraries
import os
import cv2

#initializing the directory
DATA_DIR = 'D:/my python/python code/new/data/'

if not os.path.exists(DATA_DIR):
    os.makedirs(DATA_DIR)

#initalizing the required varibles
number_of_classes = 36  #no of classes
dataset_size = 100      #no of samples for each class

#initializing the OpenCv webcam option
cap = cv2.VideoCapture(0)
for j in range(number_of_classes):
    if not os.path.exists(os.path.join(DATA_DIR, str(j))):
        os.makedirs(os.path.join(DATA_DIR, str(j)))

    print('Collecting data for class {}'.format(j))
    done = False
while True:
```

```python
    done = False
while True:
        ret, frame = cap.read()
        cv2.putText(frame, 'Ready? Press "R" ! :)', (100, 50), cv2.FONT_HERSHEY_SIMPLEX, 1.3, (0, 255, 0), 3,
                    cv2.LINE_AA)
        cv2.imshow('frame', frame)
        if cv2.waitKey(25) == ord('q'):
            break
counter = 0
while counter < dataset_size:
        ret, frame = cap.read()
        cv2.imshow('frame', frame)
        cv2.waitKey(25)
        cv2.imwrite(os.path.join(DATA_DIR, str(j), '{}.jpg'.format(counter)), frame)
        counter += 1

#realesing the web cam
cap.release()
cv2.destroyAllWindows()
```

Now with all these samples we need the specific numbers that will represent the sign language symbol in an image or in other words we need the X,Y coordinates of fingers that are different for different symbol in an image for each sample image.

For example: In the below pictures each dot has its own significance and each dot has x, y, z, coordinates.





So ,from each sample image we collect these x,y coordinates as data and their respective class as labels and save it in pickle file to train our classifier.

```
#to process data and prepare it for classifier
import os
import pickle
import mediapipe as mp # library that supports hand tracking
import cv2
import matplotlib.pyplot as plt

mp_hands = mp.solutions.hands
mp_drawing = mp.solutions.drawing_utils
mp_drawing_styles = mp.solutions.drawing_styles
hands = mp_hands.Hands(static_image_mode=True, min_detection_confidence=0.3)

DATA_DIR = 'D:/my python/python code/new/data/'
data = []
labels = []
for dir_ in os.listdir(DATA_DIR):
    for img_path in os.listdir(os.path.join(DATA_DIR, dir_)):
        data_aux = []
        x_ = []
        y_ = []
        img = cv2.imread(os.path.join(DATA_DIR, dir_, img_path))
        img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        results = hands.process(img_rgb)
        if results.multi_hand_landmarks:
            for hand_landmarks in results.multi_hand_landmarks:
                for i in range(len(hand_landmarks.landmark)):
                    x = hand_landmarks.landmark[i].x
                    y = hand_landmarks.landmark[i].y

                    x_.append(x)
                    y_.append(y)
```

```
            for i in range(len(hand_landmarks.landmark)):
                x = hand_landmarks.landmark[i].x
                y = hand_landmarks.landmark[i].y
                data_aux.append(x - min(x_))
                data_aux.append(y - min(y_))

        data.append(data_aux)
        labels.append(dir_)

f = open('data.pickle', 'wb')
pickle.dump({'data': data, 'labels': labels}, f)
f.close()
```

data.pickle is a dictionary file that contains-

data: "X and Y coordinates for each single image of our sample data".

Labels: "label representing class of each image in sample data

3.3 Selection of Classifier:

In order to select the best classifier for your dataset, we can evaluate the accuracy scores of different classifiers using Scikit-learn. Scikit-learn provides various classifiers, including RandomForestClassifier, LogisticRegression, MultinomialNB, MLPClassifier, DecisionTreeClassifier, SVC, and AdaBoostClassifier.

To find the accuracy score, we would train each classifier on our dataset and then evaluate its performance by comparing the predicted labels with the actual labels in our data. The accuracy score represents the proportion of correctly classified instances.

By calculating and comparing the accuracy scores of different classifiers, we can identify which classifier performs the best on our dataset. This approach helps us to determine the most suitable classifier for our sign language symbol recognition task without relying on plotting or visual analysis, which can be challenging due to the large size of the dataset.

```python
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.ensemble import AdaBoostClassifier
from sklearn import metrics
import pandas as pd
import pickle
from sklearn.model_selection import train_test_split
import numpy as np

data_dict=pickle.load(open('./data.pickle','rb'))

data=np.asarray(data_dict['data'])
labels=np.asarray(data_dict['labels'])

x_train,x_test,y_train,y_test=train_test_split(data,labels,test_size=0.2,shuffle=True,stratify=labels)

results_1 = pd.DataFrame(columns = ['Classifier','Accuracy', 'Precision', 'Recall', 'F1-Score'])

classifiers = [RandomForestClassifier(),LogisticRegression(),MultinomialNB(),MLPClassifier(),
               DecisionTreeClassifier(),SVC(),AdaBoostClassifier()]
for clf in classifiers:
    # Fit the model
    clf.fit(x_train, y_train)
    #Make predictions
    y_pred = clf.predict(x_test)

```

```python
    #Make predictions
    y_pred = clf.predict(x_test)

    acc, pre, rc = metrics.accuracy_score(y_test, y_pred),metrics.precision_score(y_test, y_pred,average='weighted'),
    metrics.recall_score(y_test, y_pred,average='weighted')

    f1 = metrics.f1_score(y_test, y_pred,average='weighted')
    results_1 = results_1.append({'Classifier':str(clf)[:-2],'Accuracy':acc, 'Precision':pre, 'F1-Score':f1,
                                  'Recall':rc}, ignore_index = True)
```

Output:

	Classifier	Accuracy	Precision	Recall	F1-Score
0	RandomForestClassifier	1.000000	1.000000	1.000000	1.000000
1	LogisticRegression	1.000000	1.000000	1.000000	1.000000
2	MultinomialNB	0.997183	0.997317	0.997183	0.997162
3	MLPClassifier	1.000000	1.000000	1.000000	1.000000
4	DecisionTreeClassifier	0.997183	0.997317	0.997183	0.997162
5	SVC	1.000000	1.000000	1.000000	1.000000
6	AdaBoostClassifier	0.140845	0.112230	0.140845	0.113725

From picture we can see all the classifier are good in classifying the data So with this I have go with the Random Forest classifier to train my data.

3.4 Training of Random Forest Classifier :

Training a machine learning (ML) model involves teaching the model to learn from a dataset so that it can make accurate predictions or decisions on new, unseen data. The training process typically includes the following steps:

- ❖ **Importing the classifier:** We use a classifier from Scikit-learn, which is a popular machine learning library.
- ❖ **Loading the dataset:** The dataset is loaded from a pickle file, which contains the input features and their corresponding labels.
- ❖ **Splitting the dataset:** We split the dataset into two parts: the training set and the test set. The training set is used to train the model, while the test set is used to evaluate its performance.
- ❖ **Initializing the classifier:** We initialize the chosen classifier, which sets up its initial configuration.
- ❖ **Fitting the data:** We feed the training data into the classifier to start the training process. The classifier learns from the input features and their corresponding labels.
- ❖ **Predicting and validating:** We use the trained model to predict labels for the test data and evaluate its performance by comparing the predicted labels with the actual labels.
- ❖ **Checking accuracy score:** We calculate the accuracy score, which measures how well the model performs by comparing the predicted labels with the actual labels in the test set.
- ❖ **Saving the model:** If the model performs well, we save it for future use so that we do not have to retrain it every time we want to make predictions.

```python
#importing libraries
import pickle
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import numpy as np

#loading the data file
data_dict=pickle.load(open('./data.pickle','rb'))
data=np.asarray(data_dict['data'])
labels=np.asarray(data_dict['labels'])

#splitting the data set into train and test
x_train,x_test,y_train,y_test=train_test_split(data,labels,test_size=0.2,shuffle=True,
                                               stratify=labels)
#initiallizing the classifier
classifier=RandomForestClassifier()
classifier.fit(x_train,y_train)

#predictiing to check accuracy
y_predict=classifier.predict(x_test)

#checking accuracy
score=accuracy_score(y_predict,y_test)
print('{}% of samples were classified correctly !'.format(score * 100))

#Saving the model for further use
f = open('classifier.p', 'wb')
pickle.dump({'classifier': classifier}, f)
f.close()
```

Output:

```
PS D:\my python\python code\new> & C:/Users/hp/AppData/Local/Programs/Python/Python310/python.exe "d:/my python/python code/new/model.py"
100.0% of samples were classified correctly !
PS D:\my python\python code\new>
```

3.5 Checking working of model:

To do this we will run the model and try to predict the characters by making hand gestures using web cam .

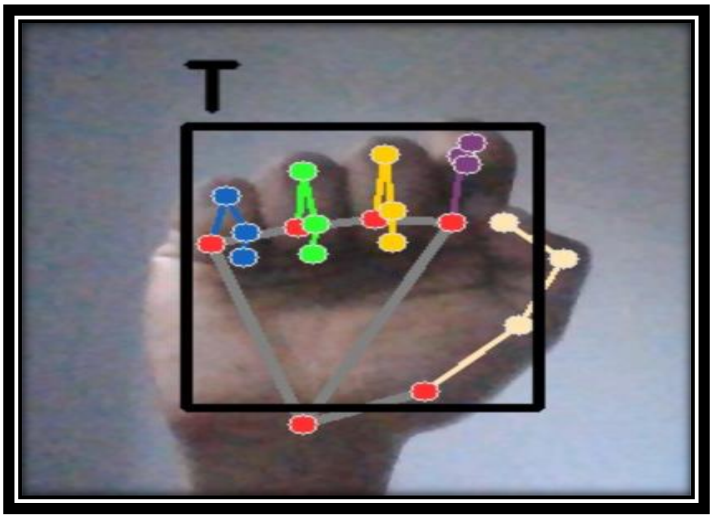
prediction.py > ...

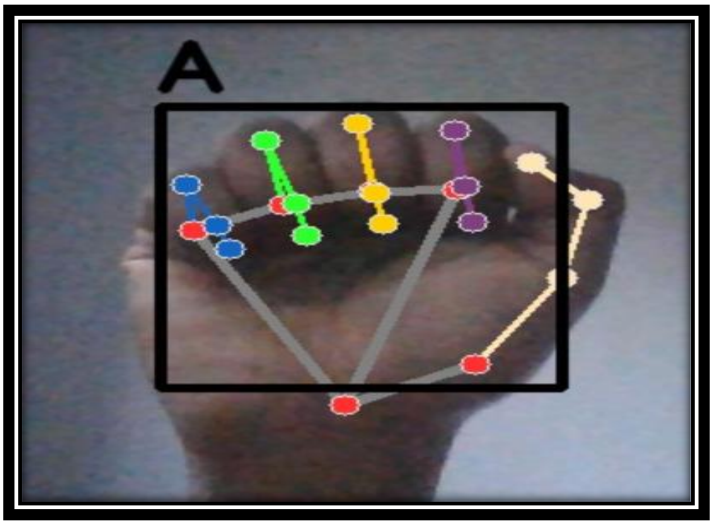
```python
#importing the libraries
import pickle
import cv2
import mediapipe as mp
import numpy as np
import enchant

#importing hte model
model_dict = pickle.load(open('./classifier.p', 'rb'))
model = model_dict['classifier']

#initalizing the webcam
cap = cv2.VideoCapture(0)

#importing all rquirements for hand tracking
mp_hands = mp.solutions.hands
mp_drawing = mp.solutions.drawing_utils
mp_drawing_styles = mp.solutions.drawing_styles
hands = mp_hands.Hands(static_image_mode=True, min_detection_confidence=0.3)

#loading dataset to get the labels
data_dict=pickle.load(open('./data.pickle','rb'))
labels=np.asarray(data_dict['labels'])

labels_dict={}
for i in labels:
    labels_dict[i]=i
```

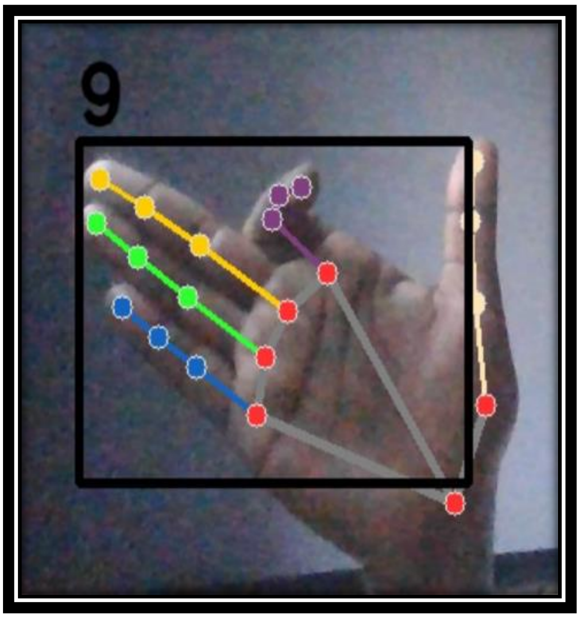
```python
while True:
    data_aux = []
    x_ = []
    y_ = []

    ret, frame = cap.read()
    if not ret:
        break
    H, W, _ = frame.shape

    frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

    results = hands.process(frame_rgb)
    if results.multi_hand_landmarks:
        for hand_landmarks in results.multi_hand_landmarks:
            mp_drawing.draw_landmarks(
                frame,  # image to draw
                hand_landmarks,  # model output
                mp_hands.HAND_CONNECTIONS,  # hand connections
                mp_drawing_styles.get_default_hand_landmarks_style(),
                mp_drawing_styles.get_default_hand_connections_style()
                )

        for hand_landmarks in results.multi_hand_landmarks:
            for i in range(len(hand_landmarks.landmark)):
                x = hand_landmarks.landmark[i].x
                y = hand_landmarks.landmark[i].y

                x_.append(x)
                y_.append(y)
```

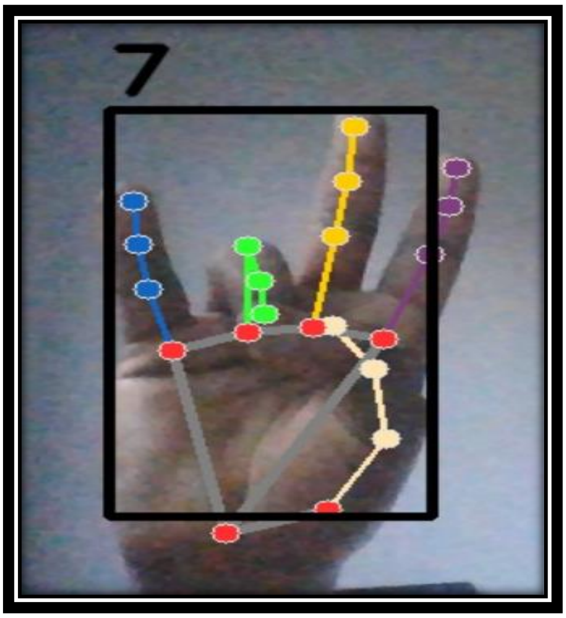
```python

            for i in range(len(hand_landmarks.landmark)):
                x = hand_landmarks.landmark[i].x
                y = hand_landmarks.landmark[i].y
                data_aux.append(x - min(x_))
                data_aux.append(y - min(y_))

        x1 = int(min(x_) * W) - 10
        y1 = int(min(y_) * H) - 10

        x2 = int(max(x_) * W) - 10
        y2 = int(max(y_) * H) - 10

        #predicting alphabet/character based on current video frame
        prediction = model.predict([np.asarray(data_aux)])
        predicted_character = labels_dict[(prediction[0])]

        #displaying the predicted character
        cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 0, 0), 4)
        cv2.putText(frame, predicted_character, (x1, y1 - 10), cv2.FONT_HERSHEY_SIMPLEX, 1.3, (0, 0, 0), 3,
                    cv2.LINE_AA)

    cv2.imshow('frame', frame)
    if cv2.waitKey(25) == ord('x'):
        break

cap.release()
cv2.destroyAllWindows()
```

Output:









3.6 Utilization of Model for making Prediction and Suggestions:

Now we will use our trained model ,other saved file and combine them using tkinter library used for GUI in python. We will use the model ,make runtime prediction and with predicted value then we make suggestion based on previous example list of words and sentence.

```python
#importing required libraries
import cv2
from string import ascii_uppercase
import tkinter as tk
from PIL import Image, ImageTk
import pickle
import mediapipe as mp
import numpy as np
import random

# Main Application :

class Application:

    def __init__(self):

        #initalizing the webcam
        self.vs = cv2.VideoCapture(0)

        self.letter=[]
        self.current_image = None

        #loading the model and ohter saved file
        self.word_list=pickle.load(open('D:/my python/python code/speech/word_list.pickle','rb'))
        self.sent_list=pickle.load(open('D:/my python/python code/speech/sent_list.pickle','rb'))
```

```python

        self.data_dict=pickle.load(open('D:/my python/python code/new/data.pickle','rb'))
        self.labels=np.asarray(self.data_dict['labels'])
        self.labels_dict={}
        for label in self.labels:
            self.labels_dict[label]=label


        self.model_dict = pickle.load(open('D:/my python/python code/new/classifier.p', 'rb'))
        self.loaded_model = self.model_dict['classifier']
        print("Loaded model from disk")

        #intializing the GUI interface for interaction
        self.root = tk.Tk()
        self.root.title("Sign Language To Text Conversion")
        self.root.protocol('WM_DELETE_WINDOW', self.destructor)
        self.root.geometry("900x800")

        self.panel = tk.Label(self.root)
        self.panel.place(x=80, y=40, width=550, height=380)

        self.T = tk.Label(self.root)
        self.T.place(x=40, y=2)
        self.T.config(text="Sign Language To Text Conversion",font=("Courier", 20, "bold"))

```

```python
        self.panel4 = tk.Label(self.root)  # Word
        self.panel4.place(x=140, y=420)

        self.T2 = tk.Label(self.root)
        self.T2.place(x=15, y=420)
        self.T2.config(text="Word :", font=("Courier", 20, "bold"))

        self.panel5 = tk.Label(self.root)  # Sentence
        self.panel5.place(x=185, y=480)

        self.T3 = tk.Label(self.root)
        self.T3.place(x=15, y=480)
        self.T3.config(text="Sentence :", font=("Courier", 20, "bold"))

        self.T4 = tk.Label(self.root)
        self.T4.place(x=165, y=535)
        self.T4.config(text="Suggestions :", fg="black",font=("Courier", 20, "bold"))

        self.bt1 = tk.Button(self.root, command=self.action, height=0, width=0)
        self.bt1.place(x=35, y=580)

        self.bt2 = tk.Button(self.root, command=self.action, height=0, width=0)
        self.bt2.place(x=335, y=580)

        self.bt3 = tk.Button(self.root, command=self.action, height=0, width=0)
        self.bt3.place(x=635, y=580)

        self.current_symbol = "Empty"
```

```
        self.current_symbol = "Empty"
        self.photo = "Empty"
        self.video_loop()

    def video_loop(self):           #function for continuous tracking of hands using webcam
        mp_hands = mp.solutions.hands
        mp_drawing = mp.solutions.drawing_utils
        mp_drawing_styles = mp.solutions.drawing_styles
        hands = mp_hands.Hands(static_image_mode=True, min_detection_confidence=0.3)

        data_aux = []
        x_ = []
        y_ = []

        #reding every frame
        ret, frame = self.vs.read()
        if ret:
            H, W, _ = frame.shape
            frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            results = hands.process(frame_rgb)

            #check if no hand gesture is made
            if not results.multi_hand_landmarks:
                self.current_image = Image.fromarray(frame_rgb)
                imgtk = ImageTk.PhotoImage(image=self.current_image)
                self.panel.imgtk = imgtk
                self.panel.config(image=imgtk)
                value=0
```

```
                imgtk = ImageTk.PhotoImage(image=self.current_image)
                self.panel.imgtk = imgtk
                self.panel.config(image=imgtk)
                value=0
                self.predict_letter(value,frame,0,0)

            #hand gesture is made
            else:
                for hand_landmarks in results.multi_hand_landmarks:
                    mp_drawing.draw_landmarks(
                        frame,  # image to draw
                        hand_landmarks,  # model output
                        mp_hands.HAND_CONNECTIONS,  # hand connections
                        mp_drawing_styles.get_default_hand_landmarks_style(),
                        mp_drawing_styles.get_default_hand_connections_style()
                        )

                for hand_landmarks in results.multi_hand_landmarks:
                    for i in range(len(hand_landmarks.landmark)):
                        x = hand_landmarks.landmark[i].x
                        y = hand_landmarks.landmark[i].y
                        x_.append(x)
                        y_.append(y)

                    for i in range(len(hand_landmarks.landmark)):
                        x = hand_landmarks.landmark[i].x
                        y = hand_landmarks.landmark[i].y
                        #getting the coordinates for prediction
```

```python
                        x = hand_landmarks.landmark[i].x
                        y = hand_landmarks.landmark[i].y
                        #getting the coordinates for prediction
                        data_aux.append(x - min(x_))
                        data_aux.append(y - min(y_))

                #getting the position of hand in the screen and adjusting as per need
                x1 = int(min(x_) * W) - 10
                y1 = int(min(y_) * H) - 10
                x2 = int(max(x_) * W) - 10
                y2 = int(max(y_) * H) - 10

                #putting a rectangle on hand to make it highlight
                frame=cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 0, 0), 4)
                #calling predict function
                self.predict_letter(data_aux,frame,x1,y1)

        self.root.after(5, self.video_loop)

    #function to predict the symbol
    def predict_letter(self,val,frame,x1,y1):

        if val == 0  :
            #if no hand gesture then symbol is blank
            self.current_symbol= " "
            if cv2.waitKey(0) == ord('q'):
                self.destructor()
```

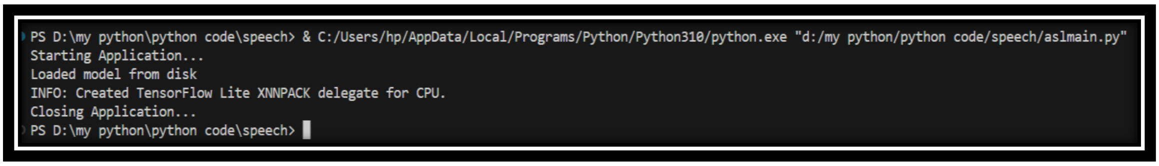
```python

        else:
            #prediction of symbol
            result = self.loaded_model.predict([np.asarray(val)])
            self.current_symbol = self.labels_dict[(result[0])]

            #putting suggested symbol to the GUI screen
            frame_rgb=cv2.putText(frame, self.current_symbol, (x1, y1 - 10), cv2.FONT_HERSHEY_SIMPLEX, 1.3, (0, 0, 0), 3,
                cv2.LINE_AA)

            #processing frame matchup the color combination
            frame_rgb = cv2.cvtColor(frame_rgb, cv2.COLOR_BGR2RGB)
            self.current_image = Image.fromarray(frame_rgb)
            imgtk = ImageTk.PhotoImage(image=self.current_image)
            #setting the pannel
            self.panel.imgtk = imgtk
            self.panel.config(image=imgtk)

        self.suggest_word() #calling function to suggest words or sentence
        return

    #function to suggest words and sentences
    def suggest_word(self):
        #synmbol is blank
        if self.current_symbol == " ":
            self.panel4.config(text=" ", font=("Courier", 30))
            self.panel5.config(text=" ", font=("Courier", 30))
            self.bt1.config(text=" ", font=("Courier", 20))
            self.bt2.config(text=" ", font=("Courier", 20))
            self.bt3.config(text=" ", font=("Courier", 20))

```

```
        elif self.current_symbol in self.labels_dict:
            #getting words and sentences from pre-saved starting with current symbol

            #list to load sentence that starts with current symbol
            word=[word for word in self.word_list if word.startswith(self.current_symbol)]
            #list to load sentence that starts with current symbol
            sent=[sent for sent in self.sent_list if sent.startswith(self.current_symbol.lower())]

            if len(sent)== 0:
                #intializing sentence empty string to avoid errors
                sent=['']

            if len(word)==0:
                #intializing random _words with empty string to avoid errors
                random_words=['','','']

            elif len(word)==1:
                #intializing random _words with word and empty string to avoid errors
                random_words=word+ ['', '']

            elif len(word)==2:
                #intializing random _words with word and empty string to avoid errors
                random_words=word+ ['']

            else:
                #picking random indices from list of words
                random_indices = random.sample(range(len(word)), 3)
                random_words = [word[index] for index in random_indices]

```

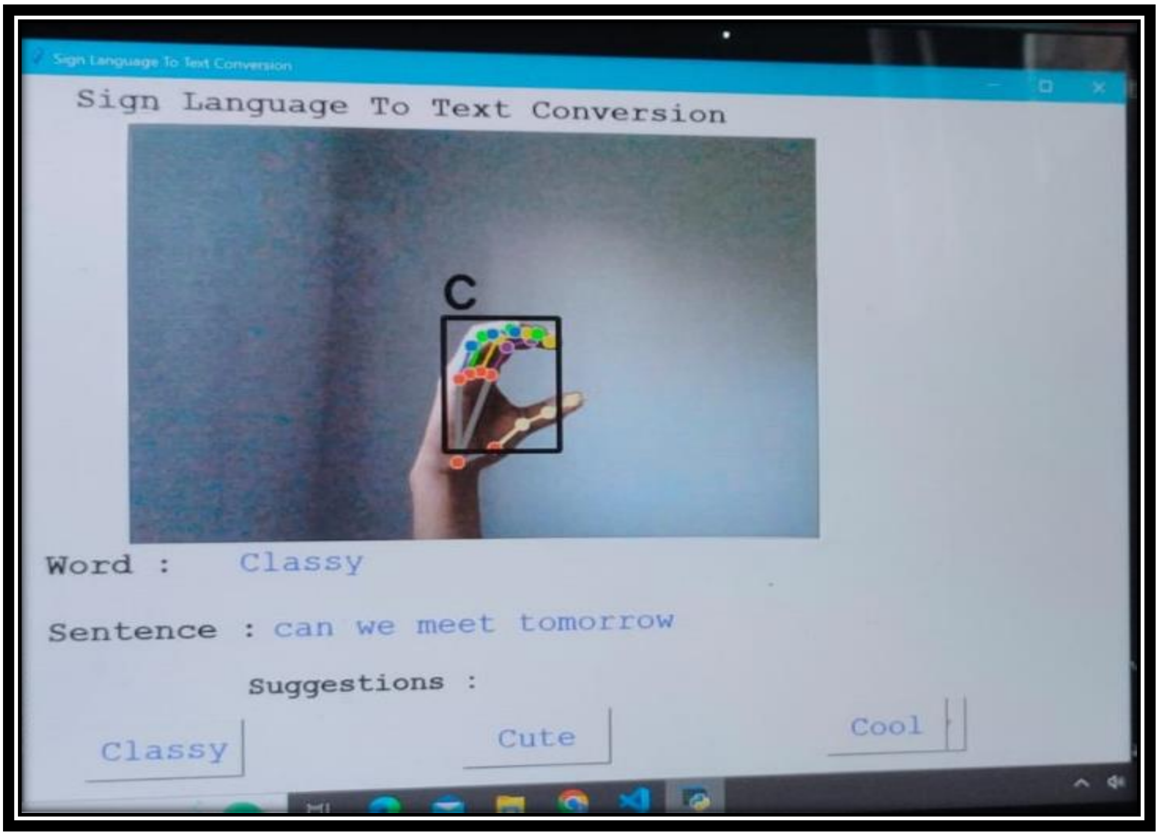
```
                random_indices = random.sample(range(len(word)), 3)
                random_words = [word[index] for index in random_indices]

            #setting the GUI Pannel with suggested words and sentences
            self.panel4.config(text=random_words[0], font=("Courier", 20))
            self.panel5.config(text=sent[0],font=("Courier", 20))
            self.bt1.config(text=random_words[0], font=("Courier", 20))
            self.bt2.config(text=random_words[1], font=("Courier", 20))
            self.bt3.config(text=random_words[2], font=("Courier", 20))
        return

    #dummy function for buttons
    def action(self):
        pass

    #function to close GUI window and release webcam feom use
    def destructor(self):
        print("Closing Application...")
        self.root.destroy()
        self.vs.release()
        cv2.destroyAllWindows()

print("Starting Application...")


#to intialize and start the application
(Application()).root.mainloop()
```

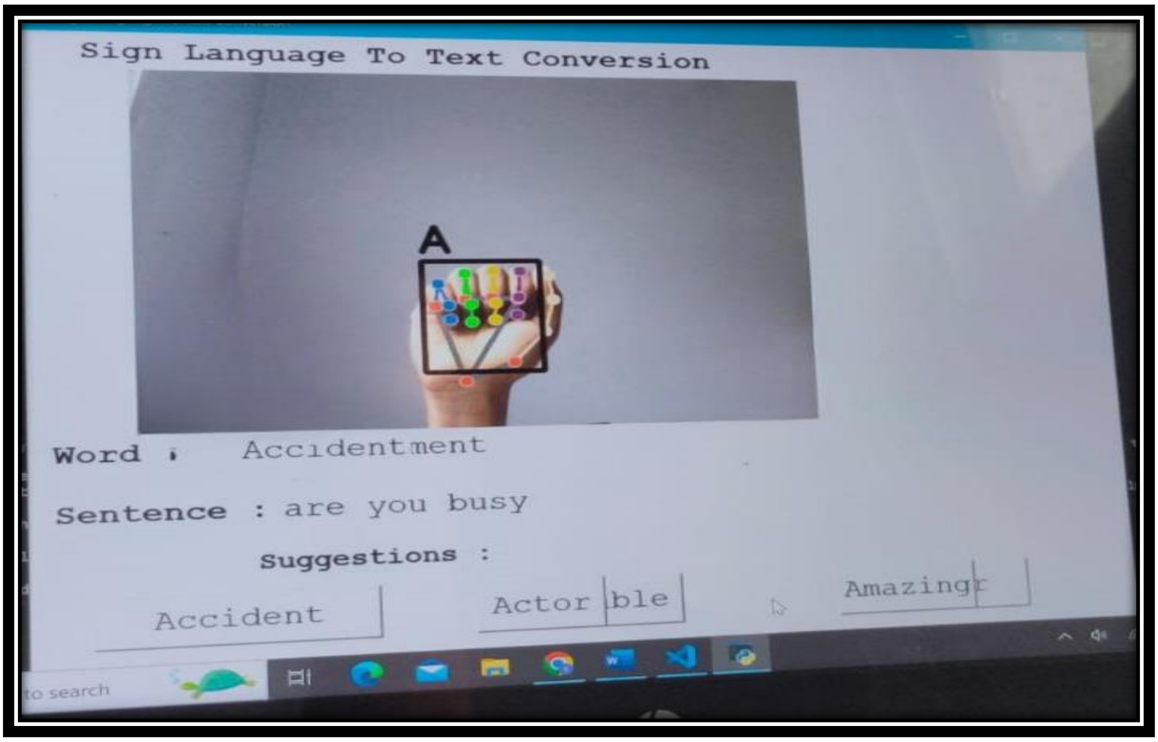
✓ 19.1s

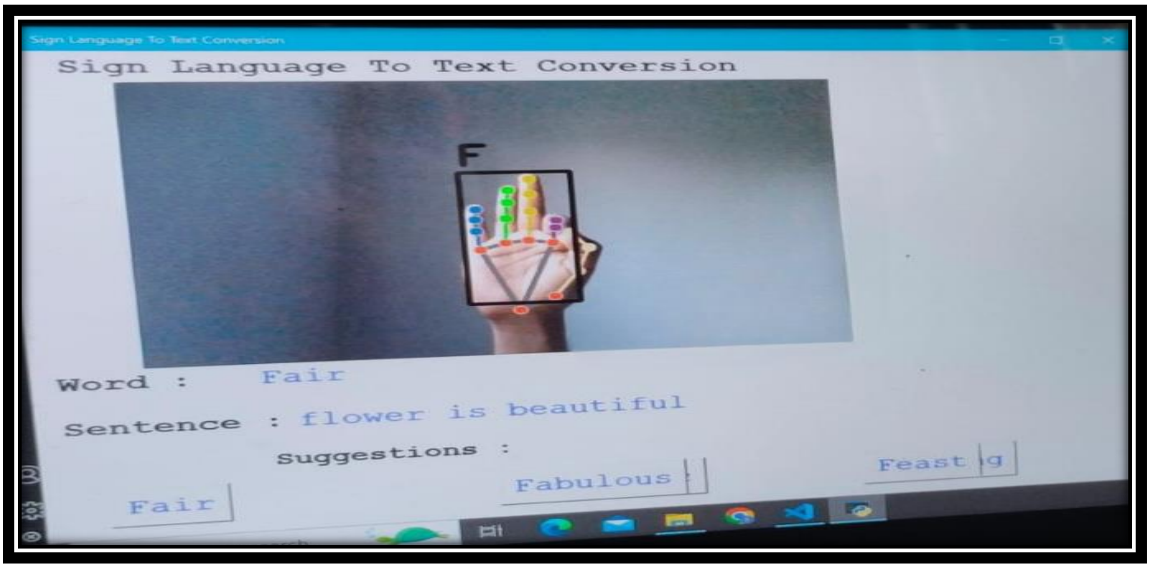
3.7 Output:

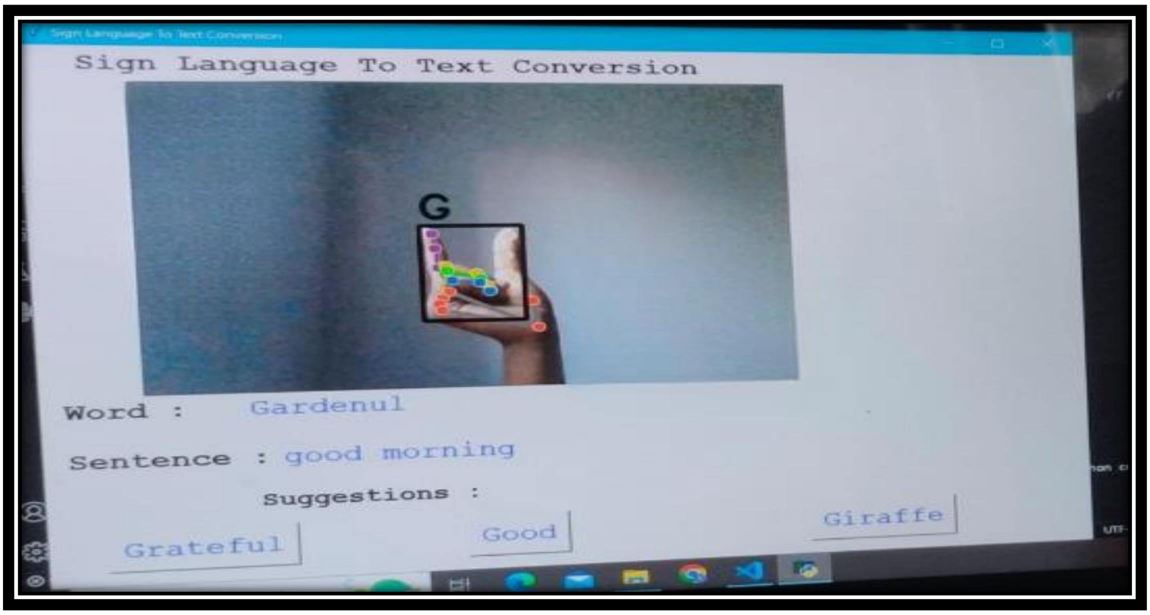


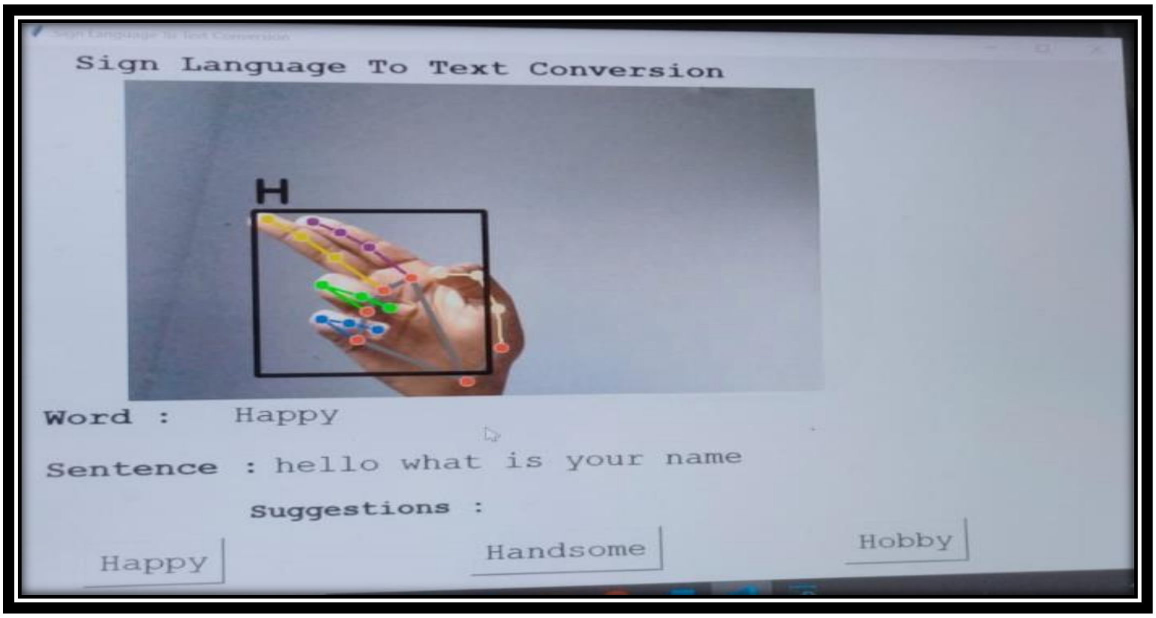
```
PS D:\my python\python code\speech> & C:/Users/hp/AppData/Local/Programs/Python/Python310/python.exe "d:/my python/python code/speech/aslmain.py"
Starting Application...
Loaded model from disk
INFO: Created TensorFlow Lite XNNPACK delegate for CPU.
Closing Application...
PS D:\my python\python code\speech>
```

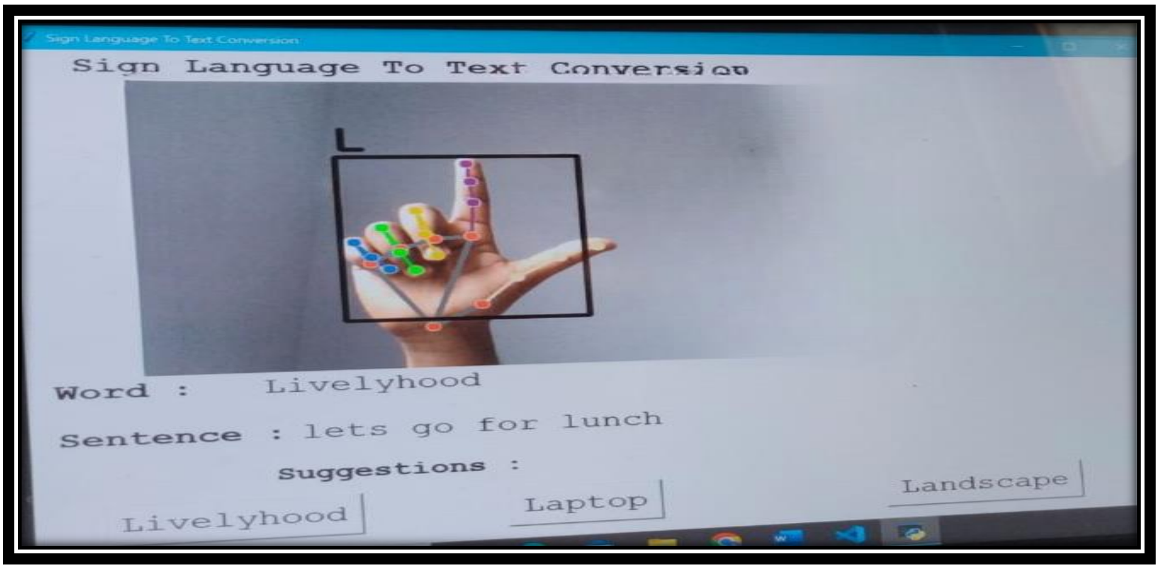


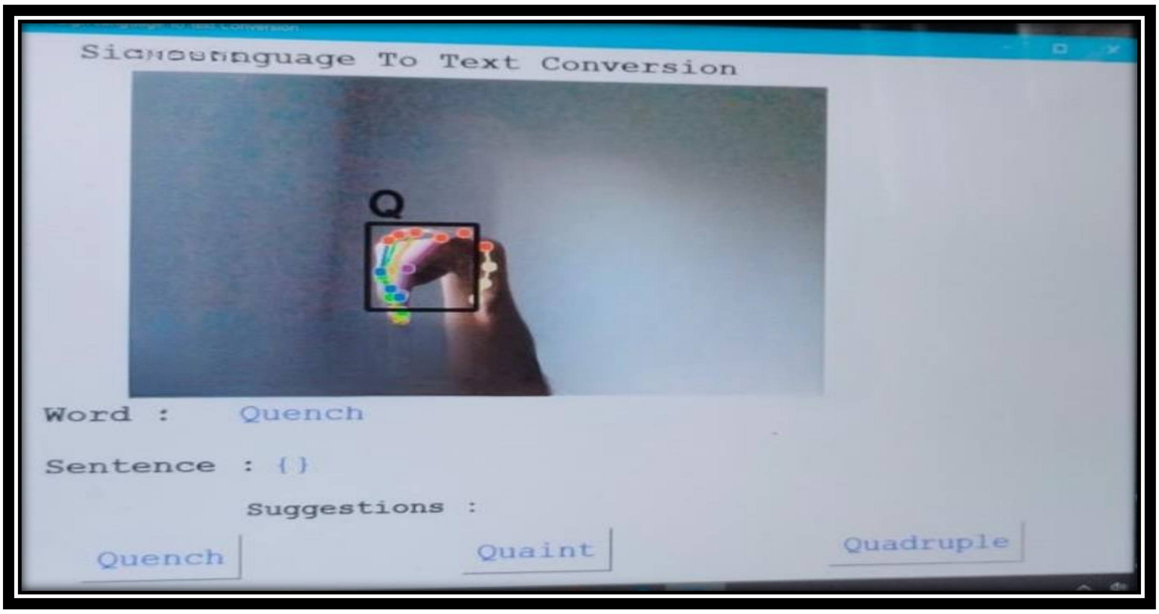


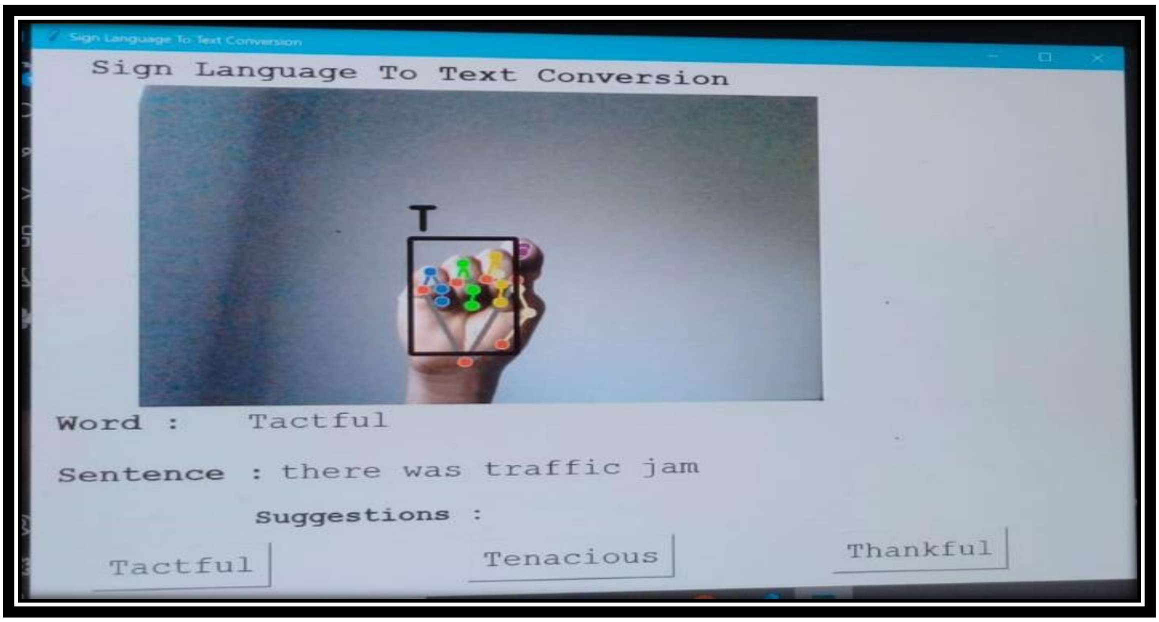

Sign Language To Text Conversion
Sign Language To Text Conversion
F
Word : Fair
Sentence : flower is beautiful
Suggestions :
Fair
Fabulous
Feast


Sign Language To Text Conversion
G
Word : Gardenul
Sentence : good morning
Suggestions :
Grateful
Good
Giraffe


Sign Language To Text Conversion
H
Word : Happy
Sentence : hello what is your name
Suggestions :
Happy
Handsome
Hobby


Sign Language To Text Conversion
L
Word : Livelyhood
Sentence : lets go for lunch
Suggestions :
Livelyhood
Laptop
Landscape


Sign Language To Text Conversion
Q
Word : Quench
Sentence : {}
Suggestions :
Quench
Quaint
Quadruple


Sign Language To Text Conversion
T
Word : Tactful
Sentence : there was traffic jam
Suggestions :
Tactful
Tenacious
Thankful

Result:

SignCompanion has demonstrated exceptional performance in converting text into sign language gesture videos and accurately predicting sign gestures for textual inputs. The intuitive and user-friendly graphical user interfaces (GUIs) developed for text-to-sign conversion and sign-to-text prediction have proven to be highly efficient. The sign-to-text classifier achieves an impressive accuracy score of 100%, showcasing its robust performance even on new data. Additionally, the system's suggestion feature operates flawlessly, providing valuable and helpful suggestions. SignCompanion's overall success highlights its potential as a powerful tool for bridging communication gaps and fostering inclusivity. With further enhancements and potential application development, SignCompanion could be made readily accessible to users, making a significant positive impact in facilitating communication between hearing-impaired individuals and the general population.

Conclusion:

In conclusion, SignCompanion offers a unique solution by providing bidirectional support, allowing both the impaired and non-impaired individuals to communicate effectively. It eliminates the need for intermediaries or additional devices, making communication more seamless. By converting text into sign language gestures and vice versa, SignCompanion addresses the challenges faced by hearing-impaired individuals in expressing themselves and understanding others. Its accuracy and efficiency have been demonstrated through the development of a trained classifier with a perfect accuracy score. SignCompanion has the potential to make a significant impact in bridging the communication gap and promoting inclusivity. With further development and modifications, it can be transformed into a user-friendly application accessible to a wide range of users, contributing to a more inclusive and accessible society.

Use Cases

The SignCompanion app holds several practical uses in real life, including:

1. **Communication Support:** SignCompanion can serve as a valuable tool for facilitating communication between hearing-impaired individuals and those who do not understand sign language. It enables seamless communication by converting text messages or spoken words into sign language gesture videos, allowing both parties to understand and interact effectively.
2. **Education and Learning:** The app can be utilized in educational settings to aid hearing-impaired students in understanding and grasping lessons. Teachers can input text-based content, which SignCompanion converts into sign language videos, enhancing the learning experience and promoting inclusivity in the classroom.
3. **Social Interactions:** SignCompanion can assist in breaking down communication barriers during social interactions. It allows individuals who are unfamiliar with sign language to communicate with hearing-impaired individuals effortlessly, fostering better understanding and connection.
4. **Accessibility in Public Spaces:** The app can be integrated into public spaces, such as airports, hospitals, or government offices, to provide sign language interpretation and assistance to hearing-impaired individuals. This promotes equal access to information and services for all individuals.
5. **Language Learning:** The app can be utilized as a tool for learning sign language. It provides visual demonstrations of sign language gestures, aiding in the acquisition and practice of sign language skills.

BIBLIOGRAPHY

- Python Programming Documentations
  https://docs.python.org/3/
- Tkinter Documentation
  https://docs.python.org/3/library/tk.html
- EasyGui Documentation
  https://easygui.readthedocs.io/en/latest/tutorial.html
- OpenCV Documentation
  https://docs.opencv.org/4.x/
- Media-pipe Hand Gesture Documentation
  https://developers.google.com/mediapipe
- Scikit-Learn Documentation
  https://scikit-learn.org/stable/auto_examples/classification/
- Speech Recognition in Python
  https://pypi.org/project/SpeechRecognition/
- ASL Sign Language Alphabet
  https://www.kaggle.com/datasets/grassknoted/asl-alphabet
- NLP Documentation
  https://www.nltk.org/
- GitHub