Machine Learning Based Sign Language Recognition

A project report Submitted
for partial completion of the degree of
Bachelor of Engineering

By

Abhinav Kashyap (Roll No. : 10331720002)
Spandan Das (Roll No. : 10331720056)
Abhinav Jha (Roll No. : 10331720001)
Ruchika (Roll No. : 10331720046)

Under the supervision of
Dr. Sabyasachi Samanta, Dept. of CSE (CS), HIT

HALDIA INSTITUTE OF TECHNOLOGY
DEPARTMENT OF CSE (CYBER SECURITY)
HALDIA, PURBA MEDINIPUR, WEST BENGAL, INDIA 2023

DECLARATION

We hereby declare that this project work titled **“Machine Learning Based Sign Language Recognition”** is my original work and no part or it has been submitted for any other degree purpose or published in any other form till date.

Signature of students

…………………………………..
Abhinav Kashyap (Roll No. : 10331720002)

……………………………………
Spandan Das (Roll No. : 10331720056)

……………………………………
Abhinav Jha (Roll No. : 10331720001)

……………………………………
Ruchika (Roll No. : 10331720046)

ACKNOWLEDGEMENT

We take this occasion to thank God, almighty for blessing us with his grace and taking our endeavor to a successful culmination. We extend our sincere and heartfelt thanks to our esteemed guide, Dr. Sabyasachi Samanta for providing us with the right guidance and advice at the crucial junctures and for showing me the right way.

We would like to thank the other faculty members also, on this occasion. Last but not least, we would like to thank our friends and family for the support and encouragement they have given us during the course of our work.

INDEX

FIGURE INDEX

ABSTRACT

This project presents a novel approach to real-time sign language recognition by leveraging statistical methods and transfer learning from a pre-trained model, specifically the MediaPipe framework. The system operates on live camera feed input, allowing for seamless integration into various applications such as communication aids for individuals with hearing impairments. The core of the system involves employing statistical methods to analyze hand gestures and recognize sign language patterns. The project utilizes transfer learning by fine-tuning a pre-trained model from MediaPipe, a robust and efficient framework for hand and pose detection. This approach significantly accelerates the model's adaptation to sign language gestures, improving its accuracy and performance.

This approach's significant advantage lies in its ability to be trained for hand sign recognition using just a single image as a dataset. This greatly improves efficiency by significantly reducing the number of images that need to be processed.

The system provides a user-friendly interface for real-time result visualization. Users can observe the recognition outcomes as they sign, fostering immediate feedback and enhancing the learning experience. The integration of the live camera feed ensures the system's responsiveness to dynamic signing gestures. The project's contributions include a streamlined sign language recognition pipeline that combines the strengths of statistical methods and transfer learning, offering a reliable and efficient solution for real-time applications. The user interface provides a practical tool for individuals, educators, and developers interested in integrating sign language recognition into various interactive environments.

Chapter 1

INTRODUCTION

1.1 Sign Language Recognition

Sign language is a complex and expressive form of communication utilized by the Deaf and hard-of-hearing communities worldwide. Unlike spoken languages, sign languages rely on gestures, facial expressions, and body movements to convey meaning. The significance of sign language goes beyond its cultural and linguistic aspects; it is a vital means of communication for individuals with hearing impairments, offering them a bridge to the world around them.

In recent years, there has been a growing interest and need for technological solutions that facilitate better communication and inclusion for the Deaf community. Sign language recognition has emerged as a promising field at the intersection of computer vision, machine learning, and assistive technology. The development of accurate and efficient systems for recognizing and interpreting sign language gestures has the potential to significantly enhance the quality of life for individuals with hearing impairments.

Traditionally, sign language recognition has faced challenges related to variability in signing styles, regional nuances, and the dynamic nature of gestures. However, with the advent of cutting-edge technologies, there is an opportunity to revolutionize the landscape of sign language recognition. Our focus lies in presenting a new method that leverages computer vision techniques and statistical techniques to enhance the accuracy and adaptability of sign language recognition systems.

As society continues its pursuit of inclusivity, understanding the transformative potential of advanced sign language recognition technology becomes imperative. Our proposed method seeks to not only address existing challenges but also pave the way for more effective and seamless communication between individuals with hearing impairments and those without. By breaking down traditional communication barriers, our approach aspires to foster an environment where everyone, irrespective of their abilities, can participate in meaningful and enriched interactions.

Hand Sign Recognition Methods:-

There are several methods to detect hand signs, such as using Deep Learning Models, Feature Extraction, Background Subtraction, 3D Image Rendering, Gesture Recognition APIs, template matching, etc.

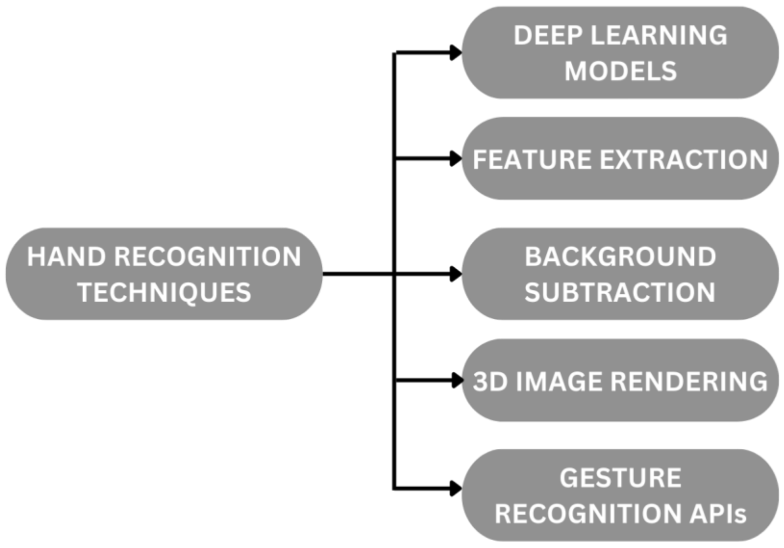


Fig. 1.1: Different Types of Hand Sign Recognition Methods

1. **Deep Learning Models:-**
   - **Convolutional Neural Networks (CNNs):** CNNs are particularly effective for image-related tasks. They can be trained to recognize hand gestures by learning hierarchical features from image data.
   - **Recurrent Neural Networks (RNNs)**: RNNs can capture temporal dependencies in sequences of hand gesture images, making them suitable for recognizing dynamic gestures.
2. **Feature Extraction:-**
   - **Histogram of Oriented Gradients (HOG**): HOG is a feature descriptor that captures the distribution of gradient orientations in an image. It can be used to represent the shape and edges of the hand in a gesture.
   - **Local Binary Patterns (LBP):** LBP is a texture descriptor that focuses on local patterns within an image, useful for capturing texture information in hand gestures.
3. **Background Subtraction:-**

Background Subtraction Algorithms: These algorithms help in isolating the hand from the background. Once the background is subtracted, hand gesture recognition can be performed on the remaining foreground.

4. **3D Image Rendering:-**

   Depth Sensors (e.g., Kinect): Depth information can be valuable for recognizing 3D hand gestures. Depth sensors can provide a three-dimensional representation of the hand's position and movements.

5. **Gesture Recognition APIs:-**

   Utilizing Pre-trained Models: Several cloud-based services such as MediaPipe offer pre-trained models for hand gesture recognition, simplifying the integration process. These APIs often leverage deep learning models.

1.2 BACKGROUND

To build our sign language recognition system, we’ll first perform hand image capturing, extract landmarks from the captured image using MediaPipe, extract the coordinates of each landmark, and then plot the curve for each sign. Using statistical techniques to predict the output:

1. **Capture Images**
2. **Extract Landmarks**
3. **Extract co-ordinates**
4. **Plot curve**

In order to apply transfer learning, we are using MediaPipe as our pre-trained model, we’ll be applying calculations on the captured image in the following ways:

- ➢ To apply *sign detection*, this detects the presence and location of a hand in an image, but does not identify it.
- ➢ After hand detection and extracting the coordinates of all the landmarks, we have to normalize the coordinates and plot curves with those coordinates for each of the signs. All this, we have to do using transfer learning.
- ➢ And in the last portion, we have to capture real-time images and repeat the above process with it, after that we will calculate the statistical error value of the captured curve with the existing generated curves. The output will be the one having the minimum error.

1.3 OBJECTIVE

The major objective of the project is to recognize the sign language used by a person and convert it to text. Those functions are listed as follows:

- Detecting hands in the images using hand detection API.
- Extracting Landmarks.
- Normalizing and plotting curves for each letter.
- Calculating the statistical error for character prediction.

After performing all the functions on the image dataset, we have to predict which character is being shown and print it on the screen.

Chapter 2

Related work

Deep Kothadiya ET. Al. [1] proposed a deep learning-based model for word recognition from gestures. Isolated Indian Sign Language (ISL) video frames are subjected to deep learning models, specifically LSTM and GRU (feedback-based learning models), to identify signs. We utilized our own dataset, IISL2020, to test the four distinct sequential combinations of LSTM and GRU (because there are two layers of LSTM and two levels of GRU). With one layer of LSTM and one layer of GRU, the suggested model achieves approximately 97% accuracy across 11 distinct indications. Communicating with those who have speech or hearing impairments may be made easier for those who are not familiar with sign language.

Aditi Deshpande ET. Al. [2] proposed a real-time approach to ASL translation using deep learning. This project's primary goal is to create a system that can recognize the American Sign Language alphabets being signed. In order to forecast the class of the hand movements, the camera records the frames of the hand being signed, passes them through a filter, and then runs them through a classifier. The suggested approach is a first step in developing a sign language interpreter to facilitate communication. The outcome is an HCI system that allows individuals to converse with D&M individuals without having any knowledge of sign language.

Refat Khan Pathan ET. Al. [3] proposed an efficient method for identifying American Sign Language (ASL) from an image dataset. Here, 24 letters from the "Finger Spelling, A" dataset have been utilized (j and z not included because they involve motion). The primary rationale behind utilizing this dataset is the intricate background of the photos, which features various locations and hues of scenes. The photos are processed as a whole for training in the first layer, and the hand landmarks are extracted in the second layer of image processing. To train these two layers, a multi-headed convolutional neural network (CNN) model has been suggested and tested using thirty percent of the dataset. Data augmentation and dynamic learning rate reduction have been utilized to get around the overfitting issue.

Aneesh Pradeep ET. Al. [4] Proposed a technology in which hand motions are recognized by Sign Language Recognition (SLR), which then produces a text or speech response for each hand gesture. There are two types of hand motions: static and dynamic. While it may be easier to recognize static hand gestures than dynamic ones, both types of recognition are crucial for human societies. This study focuses on sign language recognition using Python and OpenCV, as well as smart gloves that can detect sign language.

Leming Guo ET. Al. [5] proposed this study, where they first do empirical experiments and demonstrate that a more comprehensive training of the spatial perception module is possible with a shallow temporal aggregation module. On the other hand, local and global temporal context information in sign language cannot be effectively captured by a shallow temporal aggregation module. We suggest a cross-temporal context aggregation (CTCA) model as a solution to this problem. In particular, we construct a dual-path network with two branches for local and global temporal context sensing. To combine the two forms of context and the linguistic prior, we further create a learning target for cross-context knowledge distillation. The resulting one-branch temporal aggregation module can understand local-global temporal and semantic context thanks to the knowledge distillation. Learning the spatial perception module is made easier by this module's simple temporal perception structure.

Manikandan J ET. Al. [6] proposed that most current solutions rely on external sensors, which are expensive and inaccessible to most people. We use CNN to train the computer and OpenCV to capture images, producing text as the result. While several earlier studies provided techniques for identifying signs in part, the goal of this study was to fully accept American Sign Language, which consists of 26 letters and 10 numerals. Few ASL letters are dynamic, while the bulk are static. Therefore, this study aimed to identify static gestures from dynamic gestures by extracting information from hand and finger motions.

Ahmed Sultan ET. Al. [7] proposed in this article that the most common datasets used in the literature for the two tasks (static and dynamic datasets that are collected from different corpora) with different contents including numerical, alphabets, words, and sentences from different SLs. Along with the many preprocessing methods used prior to training and testing, it also covers the equipment needed to create these datasets. The various methods and strategies used on these datasets are contrasted in the essay. It aims to study and focus on key techniques used in vision-based approaches, such as hybrid methods and deep learning algorithms. It discusses both the vision-based and the data-gloves-based approaches. In addition, the article provides a tabular and graphical representation of different SLR techniques.

Abdellah El Zaar ET. Al. [8] proposed a convolutional neural network (CNN)-based deep learning architecture with excellent performance. The efficacy of the suggested architecture lies in its ability to accurately identify and analyze various datasets in Sign language. One of the most significant tasks that will improve the lives of deaf people by easing their everyday lives and social inclusion is the recognition of sign language. Our approach exceeds the state-of-the-art methods with a recognition rate of 99% for ASL and ISL and 98% for ArASL. It was trained and evaluated on three datasets: one each for Arabic Sign Language Alphabet (ArASL), Irish Sign Alphabets (ISL), and American Sign Language (ASL).

CHAPTER 3

CAPTURING DATASET FOR PREPROCESSING

3.1: Tools Used:

1. **OpenCV:**

OpenCV-Python is a library of Python bindings designed to solve computer vision problems. OpenCV-Python makes use of NumPy, which is a highly optimized library for numerical operations with MATLAB-style syntax. All the OpenCV array structures are converted to and from Numpy arrays. This also makes it easier to integrate with other libraries that use Numpy such as SciPy and Matplotlib.

Mainly we have used the OpenCV library for reading images(cv2.imread(image_path)), to convert BGR TO RGB format since the input of OpenCV is BGR and we needed the output in RGB.

2. **OS**

The OS module in Python provides functions for interacting with the operating system. OS comes under Python's standard utility modules. This module provides a portable way of using operating system-dependent functionality. The *os* and *os.path* modules include many functions to interact with the file system.

We mainly used the method **listdir()** returns a list containing the names of the entries in the directory given by path. The list is in arbitrary order. We mainly used the method to access the directory in which images were present.

3.2: HAND IMAGE CAPTURING USING OpenCV

OpenCV (Open-Source Computer Vision) is a library with functions that mainly aim for real-time computer vision. OpenCV supports Deep Learning frameworks Caffe, TensorFlow, and MediaPipe.

With OpenCV, you can perform hand detection using pre-trained deep-learning hand detector models such as MediaPipe.

We are capturing the system's camera access using OpenCV, it is known as mutual exclusion.

cap = cv2.VideoCapture(0)

Until the camera access is released, no other application and use the camera during that period.

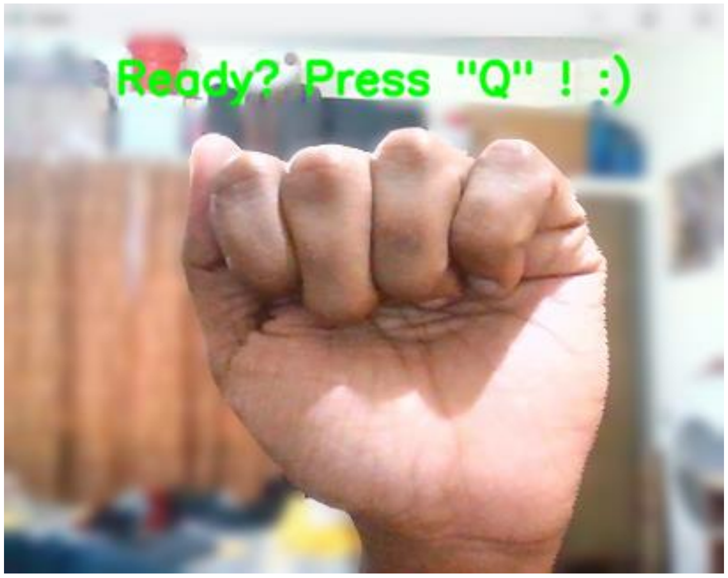


Fig. 3.1: Symbol for letter ‘A’

Then the camera access is released.

3.3: CAPTURING MULTIPLE IMAGES FOR EACH HAND:

3.3.1. Importing the necessary libraries:

```
import cv2

import os
```

3.3.2: Capturing multiple images for each sign as per ASL

We capture a number of images for each character as per the ASL (American Sign Language) format.

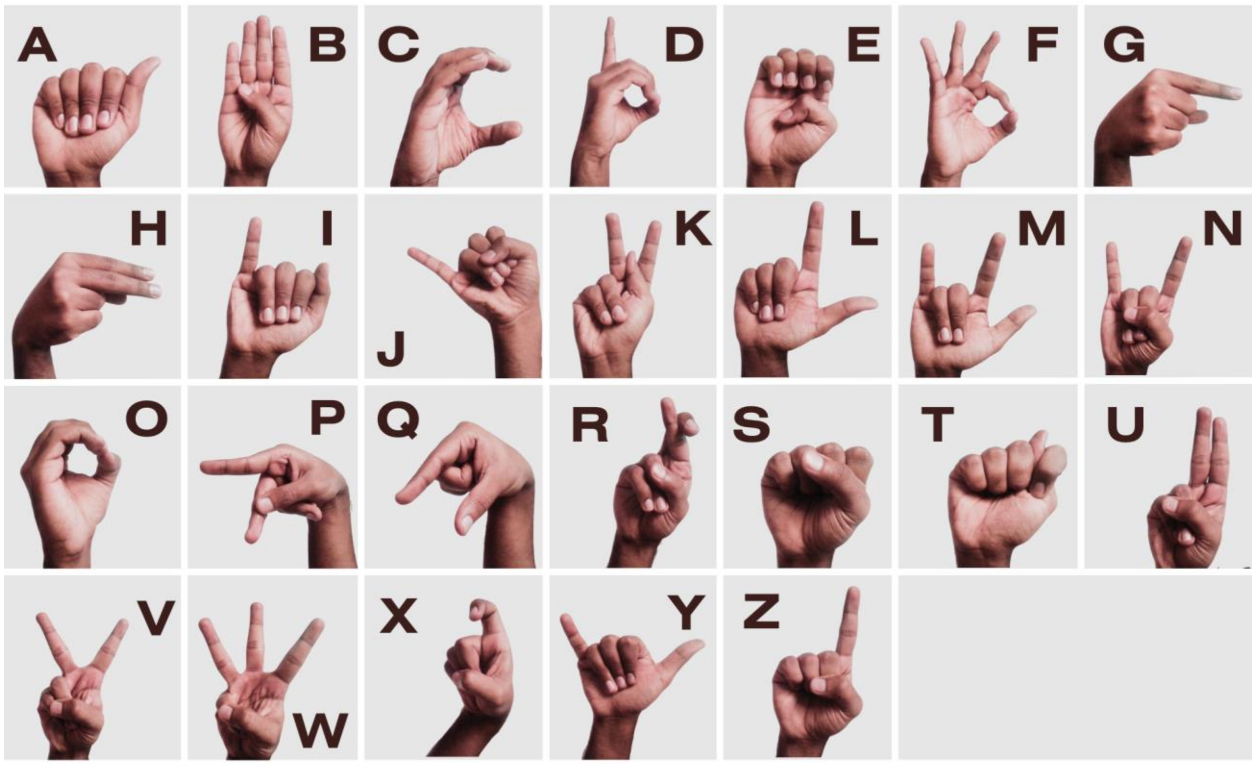


Fig 3.2: American Sign Language (ASL)

We then define the number of classes and images per class to be captured and the program captures and stores all the images in the specified path.

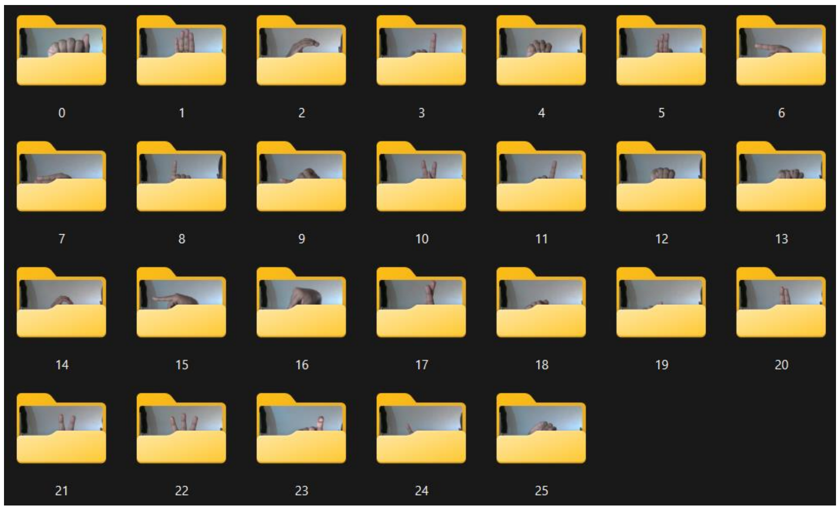


Fig 3.3: Dataset for Training

Over the duration of a few seconds, the program captures multiple images inside each directory. Each image is stored in 640 X 480 resolution with 24 bits per pixel. Before saving all the images are first converted to RGB as OpenCV captures the image in the BGR format.

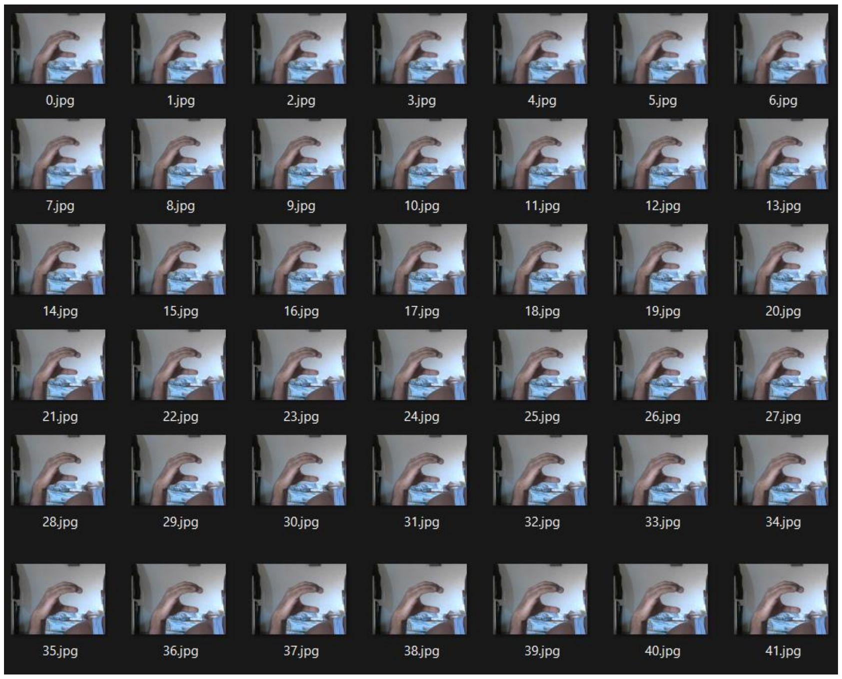


Fig 3.4: Dataset for letter 'C'

CHAPTER 4

LANDMARK DETECTION AND PLOTTING

4.1.1 Tools Used:

1. **OpenCV:**

OpenCV-Python is a library of Python bindings designed to solve computer vision problems. OpenCV-Python makes use of NumPy, which is a highly optimized library for numerical operations with MATLAB-style syntax. All the OpenCV array structures are converted to and from NumPy arrays. This also makes it easier to integrate with other libraries that use NumPy such as SciPy and Matplotlib.

Mainly we have used the OpenCV library for reading images (cv2.imread (image path)) , to convert BGR TO RGB format since the input of OpenCV is BGR and we needed the output in RGB.

2. NumPy:

NumPy is a fundamental library for numerical computing in Python. It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays. NumPy is essential for tasks in data science, machine learning, and scientific computing, offering efficient and high-performance operations. Its versatility and integration with other Python libraries make it a cornerstone for numerical computations in various domains.

3. Matplotlib:

Matplotlib is a Python library used to create 2D graphs and plots by using Python scripts. It has a module named pyplot which makes things easy for plotting by providing features to control line styles, font properties, formatting axes, etc. It supports a very wide variety of graphs and plots namely - histograms, bar charts, power spectra, error charts, etc. It is used along with NumPy to provide an environment that is an effective open-source alternative for MATLAB. We mainly have made use of this library to view images in the Jupiter notebook (plt.imshow(image)).

4. OS:

The OS module in python provides functions for interacting with the operating system. OS comes under Python's standard utility modules. This module provides a portable way of using operating system-dependent functionality. The *os* and *os.path* modules include many functions to interact with the file system.

We mainly used the method listdir() returns a list containing the names of the entries in the directory given by path. The list is in arbitrary order. We mainly used the method to access the directory in which images were present.

5. MediaPipe:

MediaPipe is an open-source library developed by Google that facilitates the building of applications for real-time perception, particularly in computer vision and machine learning. It offers pre-trained models and a modular pipeline for tasks like facial recognition, hand tracking, and pose estimation. MediaPipe simplifies complex vision tasks with a user-friendly API, enabling developers to integrate powerful perception capabilities into applications effortlessly. Its versatility makes it suitable for various projects, ranging from augmented reality to gesture recognition, advancing innovation in human-computer interaction and visual understanding applications.

4.1.2 Importing the necessary libraries

```
import os

import cv2

import matplotlib.pyplot as plt

import numpy as np

import mediapipe as mp

from math import dist, sqrt, pow
```

4.2 Hand Landmark Detection with MediaPipe

Hand landmark detection involves identifying and tracking the key points (landmarks) on a person's hand, such as the tips of the fingers and the base of the palm. Each hand contains 21 landmarks starting with index 0 on the wrist and ending with index 20 on the pinky finger’s tip.

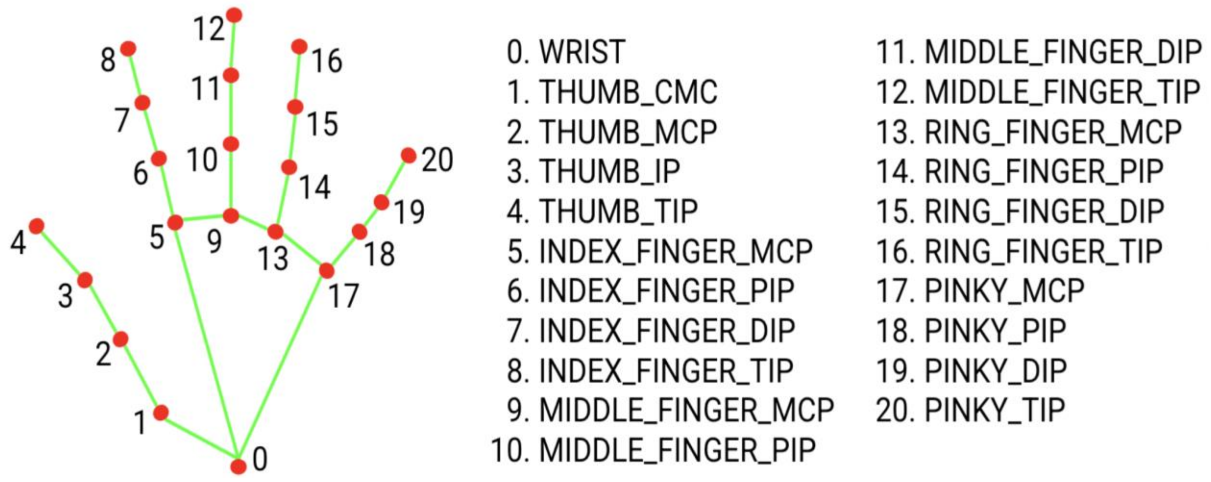


Fig 4.1: MediaPipe landmarks Index

Since running the palm detection model is time-consuming when in video or live stream running mode, Hand Landmarker uses the bounding box defined by the hand landmarks model in one frame to localize the region of hands for subsequent frames. Hand Landmarker only re-triggers the palm detection model if the hand landmarks model no longer identifies the presence of hands or fails to track the hands within the frame. This reduces the number of times Hand Landmarker triggers the palm detection model. [9]

Then we establish a MediaPipe pipeline by using the module imported and declare it’s variables as follows:

```
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=0.7, max_num_hands=1)
```

- **min_tracking_confidence-** The minimum confidence score for the hand tracking to be considered successful. This is the bounding box IoU threshold between hands in the current frame and the last frame. In the Video mode and Stream mode of Hand Landmarker, if the tracking fails, Hand Landmarker triggers hand detection. Otherwise, it skips the hand detection.
- **Static_image_mode-** It will process the image **image.jpg** and print the hand landmarks to the console.
- **max_num_hands-** This argument will indicate the maximum number of hands that the model will detect at one instance.

4.3 Fetching the Co-ordinates of each landmark

First we process the image captured by camera using **results=hands.process(img_rgb)** then we extract the landmarks from inside the captured image

```
for id, lm in enumerate(myHand.landmark):
    height, width, _ = img.shape
    temp[id]=[lm.x,lm.y]
```

Here **temp** is a dictionary that contains the X and Y coordinates of each landmark along with their indexes.

4.4 Normalizing the distance between landmarks

We are in the process of computing the distances between various landmarks with the aim of establishing a distinctive pattern for each letter. This pattern will serve as the foundational element for our predictive model, allowing us to discern and predict the spatial relationships between landmarks associated with different letters.

Function for distance calculation:

```
def distance_calculation(temp):
    res=[]
    landmark_set=[(0,8),(0,12),(0,16),(0,20),(0,4),(1,4),(5,8),(9,12),(13,16),(17,20),
                  (8,12),(8,16),(8,20),(4,8),(4,20),(4,12),(4,6),(4,10),(4,14),(4,18)]
    for m,n in landmark_set:
        res.append(dist(temp[m],temp[n]))
    return np.array(res)
```

Now we normalize the distance values obtained for the data preprocessing pipeline to convert the results in a comparable scale. In the context of calculating distances between landmarks for a prediction model, normalizing the values associated with these landmarks can contribute to the stability, efficiency, and overall effectiveness of the subsequent modeling process.

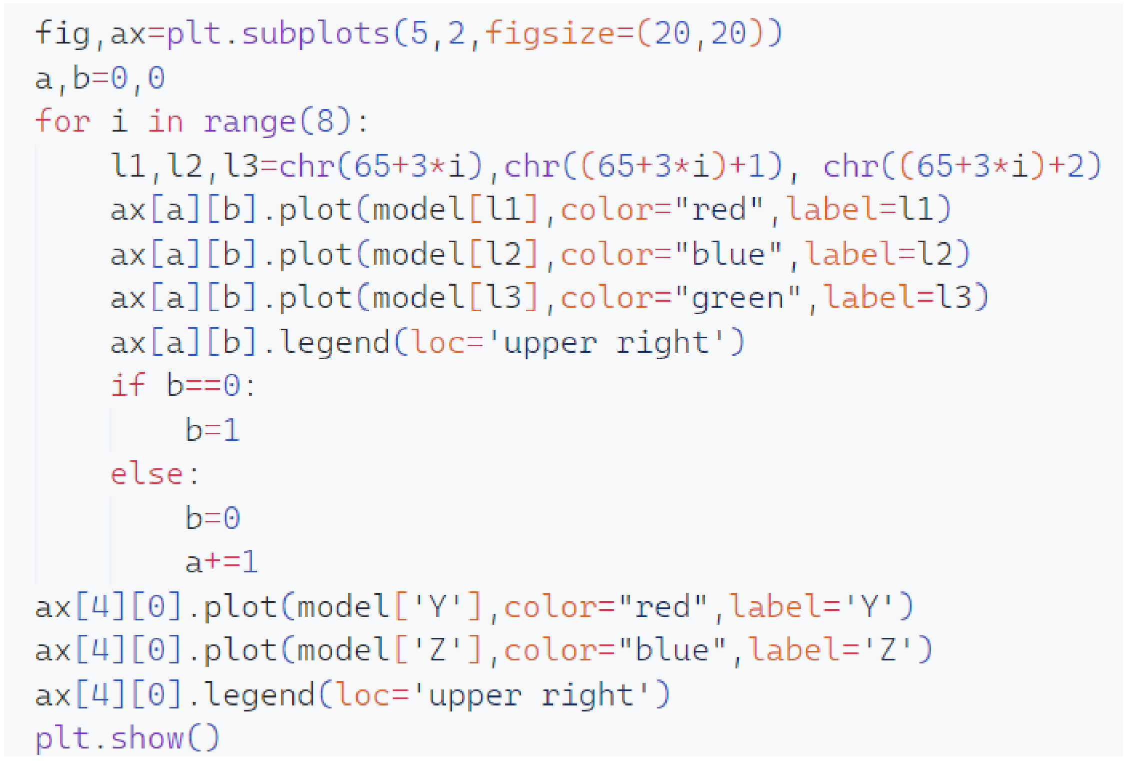
```
temp_y=distance_calculation(temp)
temp_y/=max(temp_y)
```

To obtain the normalized distance we divide each value with the maximum distance in the array.

4.5 Using Matplotlib to plot the curve for each hand sign

Matplotlib is a popular Python plotting library, we use it to visualize the curves associated with each sign. Visualizing the curves helps us **visualize patterns and Trends, Comparing Multiple Data Series, Communication, and Presentation.** Overall, plotting curves is a fundamental step in the data analysis and visualization process. It enhances the interpretability of data, facilitates decision-making, and contributes to a deeper understanding of the information at hand.

To plot the curves we use the following function:



```python
fig,ax=plt.subplots(5,2,figsize=(20,20))
a,b=0,0
for i in range(8):
    l1,l2,l3=chr(65+3*i),chr((65+3*i)+1), chr((65+3*i)+2)
    ax[a][b].plot(model[l1],color="red",label=l1)
    ax[a][b].plot(model[l2],color="blue",label=l2)
    ax[a][b].plot(model[l3],color="green",label=l3)
    ax[a][b].legend(loc='upper right')
    if b==0:
        b=1
    else:
        b=0
        a+=1
ax[4][0].plot(model['Y'],color="red",label='Y')
ax[4][0].plot(model['Z'],color="blue",label='Z')
ax[4][0].legend(loc='upper right')
plt.show()
```

We obtain the following result:

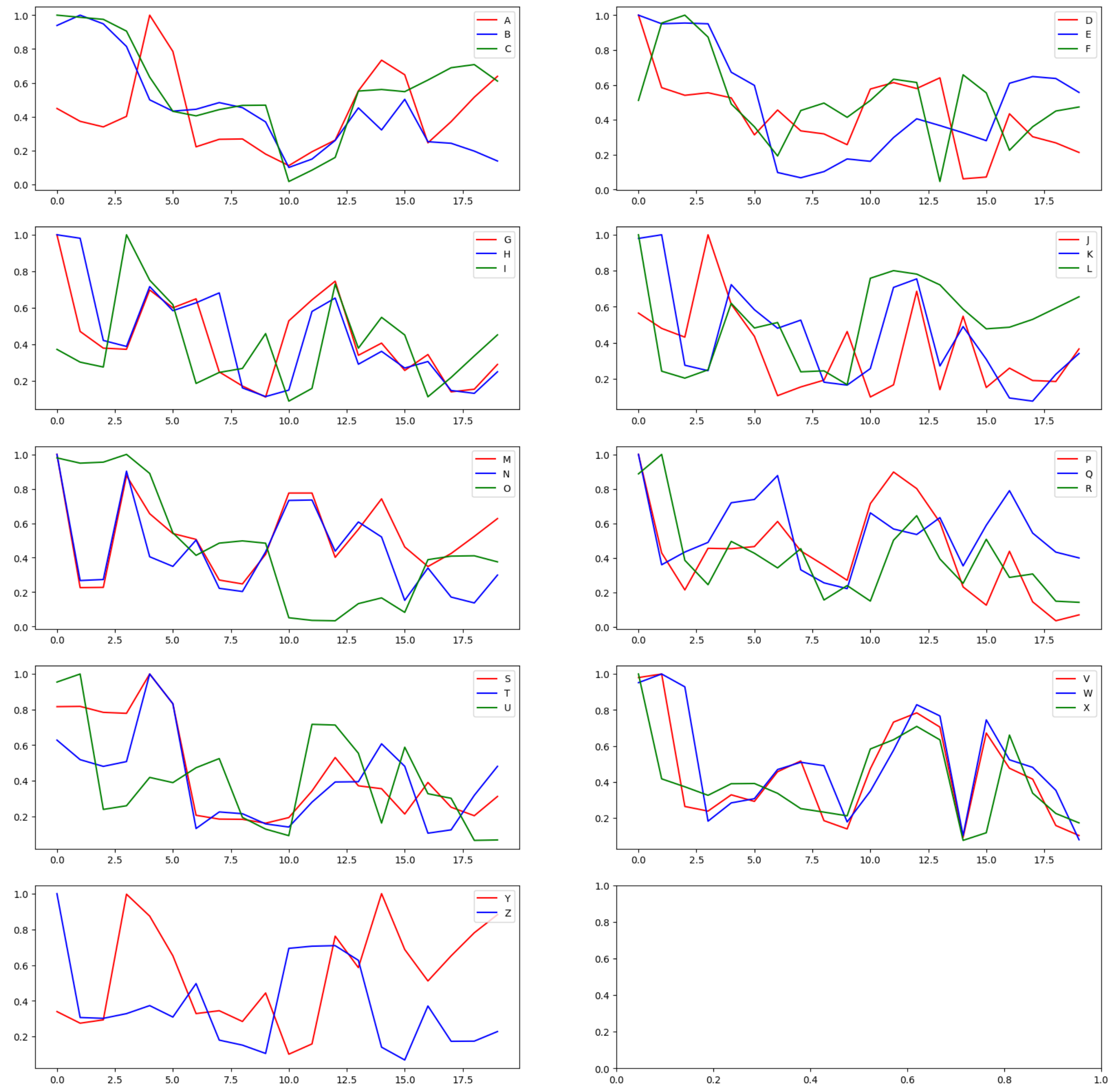
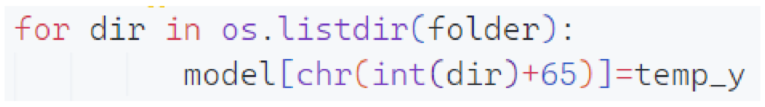


Fig 4.2: Curve for each letter of ASL

From the above curves, we can see that each letter has its own unique curves. This distinction will be used to predict the character in real-time.

4.6 Saving the model in a dictionary

By using the following code:



```
for dir in os.listdir(folder):
        model[chr(int(dir)+65)]=temp_y
```

We save all the distance data arrays with their corresponding letter in a dictionary.

CHAPTER 5
CAPTURING HAND SIGN AND PREDICTING THE CHARACTER

Utilizing **statistical methodologies**, we will compute the **Root Mean Square (RMS) error** for real-time images when processed through our pre-existing model, concurrently employing the model to predict the corresponding characters. The RMS error serves as a comprehensive metric, offering a measure of the overall differences between the predicted characters generated by our model and the actual characters represented in the real-time images.

This analytical approach involves the calculation of the square root of the average of the squared errors between the predicted and true character values. The resulting RMS error provides a holistic representation of the model's predictive performance, encapsulating both the magnitude and direction of deviations from the ground truth.

5.1. Capturing Real-time hand sign:

By using OpenCV we are gaining access to the camera.

cap = cv2.VideoCapture(0)

When dealing with multiple cameras, the number inside the parentheses indicates the specific camera employed for capturing. Within an infinite loop, we initiate the process of capturing the live feed.

A video essentially comprises a rapid succession of images. When 24 images are cycled within one second, it transforms into a video. Following this rationale, we capture multiple images per second and process them sequentially. OpenCV captures images in the BGR format by default, requiring conversion to RGB format for further processing.

The capturing and color conversion is done as follows:

```
_, frame = cap.read()

H, W, _ = frame.shape

test_img_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
test_results=hands.process(test_img_rgb)
```

5.2: Capturing the coordinates of landmarks from captured images:

As we capture images from the live feed, the acquisition rate exceeds 30 images per second. Consequently, we must undertake the task of converting the color space for each of these captured images. This process involves transforming the default color representation (BGR) into the desired color space, such as RGB, to ensure compatibility and facilitate further image processing. This iterative conversion is essential to maintain accuracy and consistency in the visual data obtained from the continuous stream of images.

We process each frame captured by the camera using **results=hands.process(img_rgb)** then we extract the landmarks from inside the captured image. Then each frame is compared with the model to obtain a prediction.

```
for id, lm in enumerate(myHand.landmark):
    height, width, _ = img.shape
    temp[id]=[lm.x,lm.y]
```

Here **temp** is a dictionary that contains the X and Y coordinates of each landmark along with their indexes.

5.3: Predicting the character from the captured image

In predicting the likely character, we adhere to the following procedural steps:

- Normalizing the distance between landmarks in each captured frame.
- Calculate the statistical error of the captured image’s curve with the curve of each of the preprocessed characters.
- The character with minimum error is selected as the predicted output.

1) To standardize the distance values derived from the data preprocessing pipeline, ensuring that the results are brought to a comparable scale, and the values are normalized. In the context of assessing distances between landmarks for a prediction model, normalizing the values linked to these landmarks enhances the stability, efficiency, and overall efficacy of the subsequent modeling procedures.

```
temp_y=distance_calculation(temp)
temp_y/=max(temp_y)
```

To obtain the normalized distance we divide each value with the maximum distance in the array.

2) There are several methods to calculate statistical error, depending on the context and the type of statistical analysis being performed. Here are a few common methods:

- **Mean Absolute Error (MAE)**
- **Mean Squared Error (MSE):**
- **Root Mean Squared (RMS):**
- **Coefficient of Determination (R-squared):**
- **Relative Absolute Error (RAE):**

In our approach, we are using **Root Mean Square (RMS)** method.

RMS measures the average error or the graphs. Here, “the average” is a point. If we simply take the sum of errors, then the result gets larger if the range of x of the graph is larger.

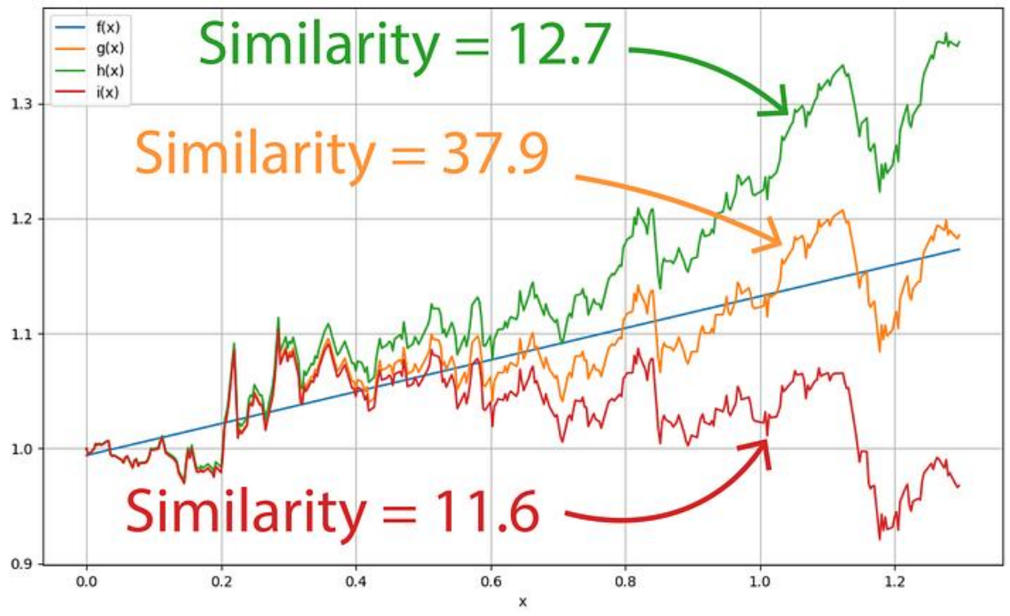


Fig 5.1: Finding the most similar curve

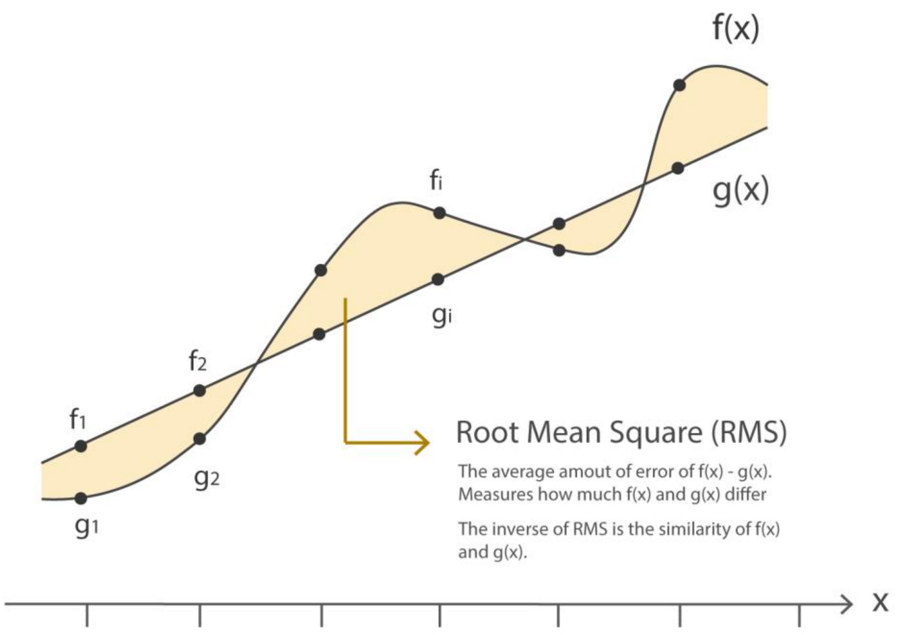

f(x)
g(x)
fi
gi
f2
g2
f1
g1
Root Mean Square (RMS)
The average amout of error of f(x) - g(x).
Measures how much f(x) and g(x) differ
The inverse of RMS is the similarity of f(x) and g(x).
x

Fig 5.2: Root Mean Square (RMS)

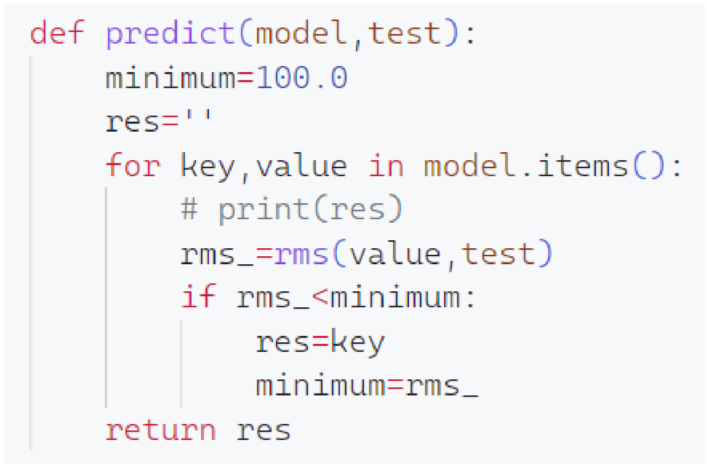
If we use the fact that RMS measures how much two graphs are different, we can use it as a measure of the graph similarity by taking the inverse of RMS. Here in this story, **we define the similarity as 1/RMS.**

The image above is an example of two graphs defined by functions f(x) and g(x). In programming or computer science, they are always discrete, meaning they are just arrays of data points f1, f2, … or g1, g2, … RMS first subtracts their graphs and then take their square. One of the reasons why we need to take their square is because a negative error must be counted as positive. After taking the average of the squared error, we apply a square root on it to cancel the side-effect of the square. [10]

We use the following function to calculate the RMS value against a calculated curve:

```
def rms(a:np.array,b:np.array)->float:
    res=0
    for i in range(20):
        res+=pow(a[i]-b[i],2)
    return sqrt(res)
```

Then finally we use the below function to predict the character:



```python
def predict(model,test):
    minimum=100.0
    res=''
    for key,value in model.items():
        # print(res)
        rms_=rms(value,test)
        if rms_<minimum:
            res=key
            minimum=rms_
    return res
```

It returns a value of char datatype, which is the predicted character.

3) Normalized coordinates from the captured frame are inputted into the model, and the Root Mean Square (RMS) is computed between these coordinates and those of all other curves. The RMS calculation serves to quantify the dissimilarity between the captured frame and each reference curve. The model then identifies the curve with the minimum RMS value, signifying the closest match or alignment. This approach ensures a robust and quantitative assessment of similarity, allowing for accurate identification of the most fitting curve amidst multiple candidates. The selection of the curve with the minimum RMS enhances the precision and reliability of the model's predictions in various applications.

Below are a few samples of the final working program:

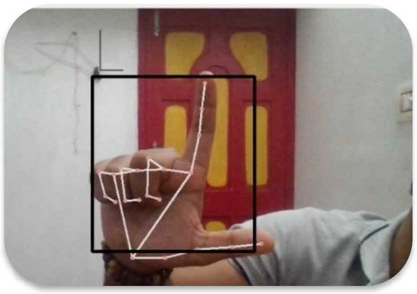


Fig 5.3.a: Output for letter 'L'

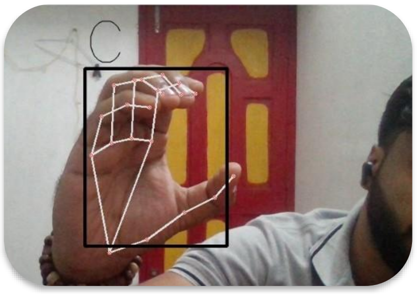


Fig 5.3.b: Output for letter 'C'

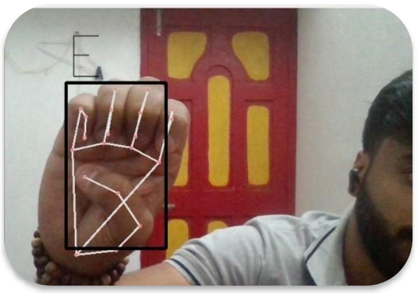


Fig 5.3.c: Output for letter 'E'

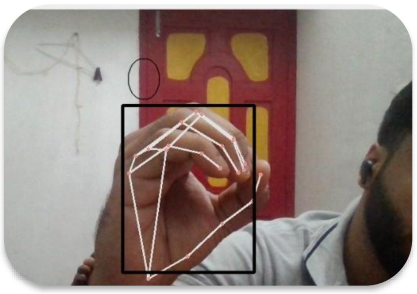


Fig 5.3.d: Output for letter 'O'

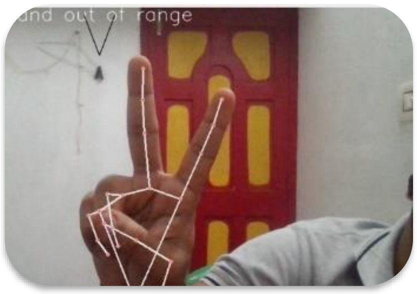


Fig 5.3.e: Output for letter 'V'

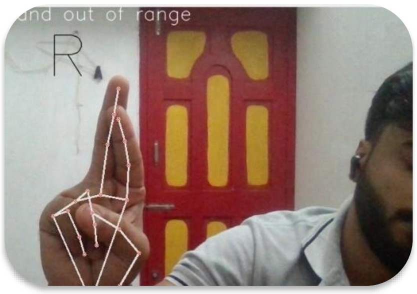


Fig 5.3.f: Output for letter 'R'

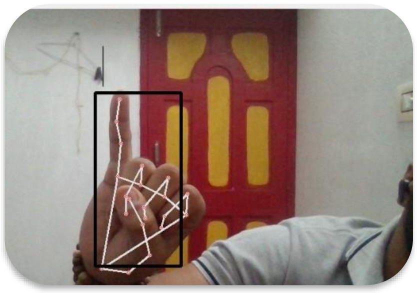


Fig 5.3.g: Output for letter 'I'

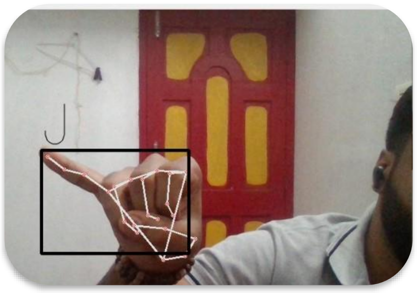


Fig 5.3.h: Output for letter 'J'

CHAPTER 6

REQUIREMENTS AND ANALYSIS

6.1 PROBLEM STATEMENT

1. **Problem of Reliability:**
2. **Problem of Accuracy:**

6.2 REQUIREMENTS SPECIFICATION

Requirement Specification

The main part of the problem is to obtain a clear understanding of the needs of what exactly is desired from the software. It is used for specifying the requirement.

6.3 SOFTWARE AND HARDWARE REQUIREMENTS

SOFTWARE & HARDWARE REQUIREMENTS

HARDWARE:

Processor:	Inter i5 or above
Memory:	8 GB RAM or above
Cache Memory:	256 KB or above
Hard Disk:	1 GB or above [at least 3 MB free space required]
Webcam:	720p or above

SOFTWARE:

1. Operating System: Windows
2. Visual Studio Code
3. Jupiter notebook

CHAPTER 7

CONCLUSION

In conclusion, this project successfully addressed the challenge of real-time sign language recognition by synergizing the capabilities of MediaPipe, OpenCV, and statistical error analysis. Leveraging the efficiency of transfer learning from a pre-trained MediaPipe model, we achieved a robust hand and pose detection system, which formed the foundation for accurate sign language recognition. The integration of OpenCV allowed for seamless live camera feed processing, enabling our system to operate in real-time. This real-time capability is crucial for practical applications, especially in scenarios where immediate feedback is essential, such as educational tools or communication aids for individuals with hearing impairments.

The incorporation of statistical error analysis played a pivotal role in refining the model's accuracy. By meticulously examining the errors in multiple signs, we enhanced the overall accuracy of the system. This methodology not only improved the model's performance but also provided insights into potential challenges and areas for future refinement. The user interface designed for this project facilitates a user-friendly experience, allowing individuals to interact with the system and observe recognition results in real-time. This immediate feedback loop contributes to a more engaging and effective learning or communication experience for users.

In summary, the amalgamation of MediaPipe, OpenCV, and statistical error analysis in our sign language recognition project has yielded a sophisticated, real-time solution. The project's success underscores the potential for technology to bridge communication gaps and empower individuals with hearing impairments. As technology continues to advance, the methodologies presented in this project pave the way for further innovations and improvements in sign language recognition systems.

BIBLIOGRAPHY

[1] Kothadiya, D., Bhatt, C., Sapariya, K., Patel, K., Gil-González, A.-B., & Corchado, J. M. (2022). Deepsign: Sign Language Detection and Recognition Using Deep Learning. In Electronics (Vol. 11, Issue 11, p. 1780). MDPI AG. https://doi.org/10.3390/electronics11111780

[2] Deshpande, A., Shriwas, A., Deshmukh, V., & Kale, S. (2023). Sign Language Recognition System using CNN. In 2023 International Conference on Intelligent and Innovative Technologies in Computing, Electrical and Electronics (IITCEE). 2023 International Conference on Intelligent and Innovative Technologies in Computing, Electrical and Electronics (IITCEE). IEEE. https://doi.org/10.1109/iitcee57236.2023.10091051

[3] Pathan, R. K., Biswas, M., Yasmin, S., Khandaker, M. U., Salman, M., & Youssef, A. A. F. (2023). Sign language recognition using the fusion of image and hand landmarks through multi-headed convolutional neural network. In Scientific Reports (Vol. 13, Issue 1). Springer Science and Business Media LLC. https://doi.org/10.1038/s41598-023-43852-x

[4] Pradeep, A., Asrorov, M., & Quronboyeva, M. (2023). Advancement Of Sign Language Recognition Through Technology Using Python And OpenCV. In 2023 7th International Multi-Topic ICT Conference (IMTIC). 2023 7th International Multi-Topic ICT Conference (IMTIC). IEEE. https://doi.org/10.1109/imtic58887.2023.10178445

[5] Guo, L., Xue, W., Guo, Q., Liu, B., Zhang, K., Yuan, T., & Chen, S. (2023). Distilling Cross-Temporal Contexts for Continuous Sign Language Recognition. In 2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). 2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). IEEE. https://doi.org/10.1109/cvpr52729.2023.01037

[6] J, M., Krishna, B. V., S, S. N., & K, S. (2022). Sign Language Recognition using Machine Learning. In 2022 International Conference on Innovative Computing, Intelligent Communication and Smart Electrical Systems (ICSES). 2022 International Conference on Innovative Computing, Intelligent Communication and Smart Electrical Systems (ICSES). IEEE. https://doi.org/10.1109/icses55317.2022.9914155

[7] Sultan, A., Makram, W., Kayed, M., & Ali, A. A. (2022). Sign language identification and recognition: A comparative study. In Open Computer Science (Vol. 12, Issue 1, pp. 191–210). Walter de Gruyter GmbH. https://doi.org/10.1515/comp-2022-0240

[8] El Zaar, A., Benaya, N., & El Allati, A. (2022). Sign Language Recognition: High Performance Deep Learning Approach Applyied To Multiple Sign Languages. In K. Slimani, O. Gerasymov, M. Ait Kbir, S. Bennani Dosse, S. Bourekkadi, & A. Amrani (Eds.), E3S Web of Conferences (Vol. 351, p. 01065). EDP Sciences. https://doi.org/10.1051/e3sconf/202235101065

[9] https://developers.google.com/mediapipe/solutions/vision/hand_landmarker

[10] Calculate graph similarities with Root Mean Square (RMS). medium.com/python-data-analysis. https://medium.com/python-data-analysis/calculate-graph-similarity-with-root-mean-square-rms-a30d11dc0251