**ISL REAL-TIME DETECTION**

**INTERDISCIPLINARY REPORT**

Submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering Degree in Computer Science and Engineering

By

**POORNASHREE.E**

**(Reg.No.42110984)**

****

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**SCHOOL OF COMPUTING**

**SATHYABAMA**

**INSTITUTE OF SCIENCE AND TECHNOLOGY**

**(DEEMED TO BE UNIVERSITY)**

**CATEGORY- 1 UNIVERSITY BY UGC**

**Accredited with Grade “A++” by NAAC I Approved by AICTE**

**JEPPIAAR NAGAR, RAJIV GANDHI SALAI CHENNAI – 600119**

**APRIL - 2025**



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**BONAFIDE CERTIFICATE**

This is to certify that this Project Report is the bonafide work of **POORNASHREE.E** **(42110984)** who carried out the project entitled “**ISL REAL-TIME DETECTION**” under my supervision from January 2025 to April 2025.

**Internal Guide**

**Dr. D. Sheema Wilson, Ph.D.,**

**Head of the Department**

**Dr. L. Lakshmanan, M.E., Ph.D.,**



Submitted for Interdisciplinary Project Viva Voce Examination held on--------------------

**Internal Examiner External Examiner**

**DECLARATION**

I, **POORNASHREE.E (Reg.no – 42110984)** hereby declare that the Interdisciplinary Project Report entitled “ISL REAL-TIME DETECTION” done by me under the guidance of **Dr.D.SHEEMA WILSON.Ph.D.,** is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering degree in **Computer Science and Engineering.**

**DATE:**

**PLACE: Chennai SIGNATURE OF THE CANDIDATE**

**ACKNOWLEDGEMENT**

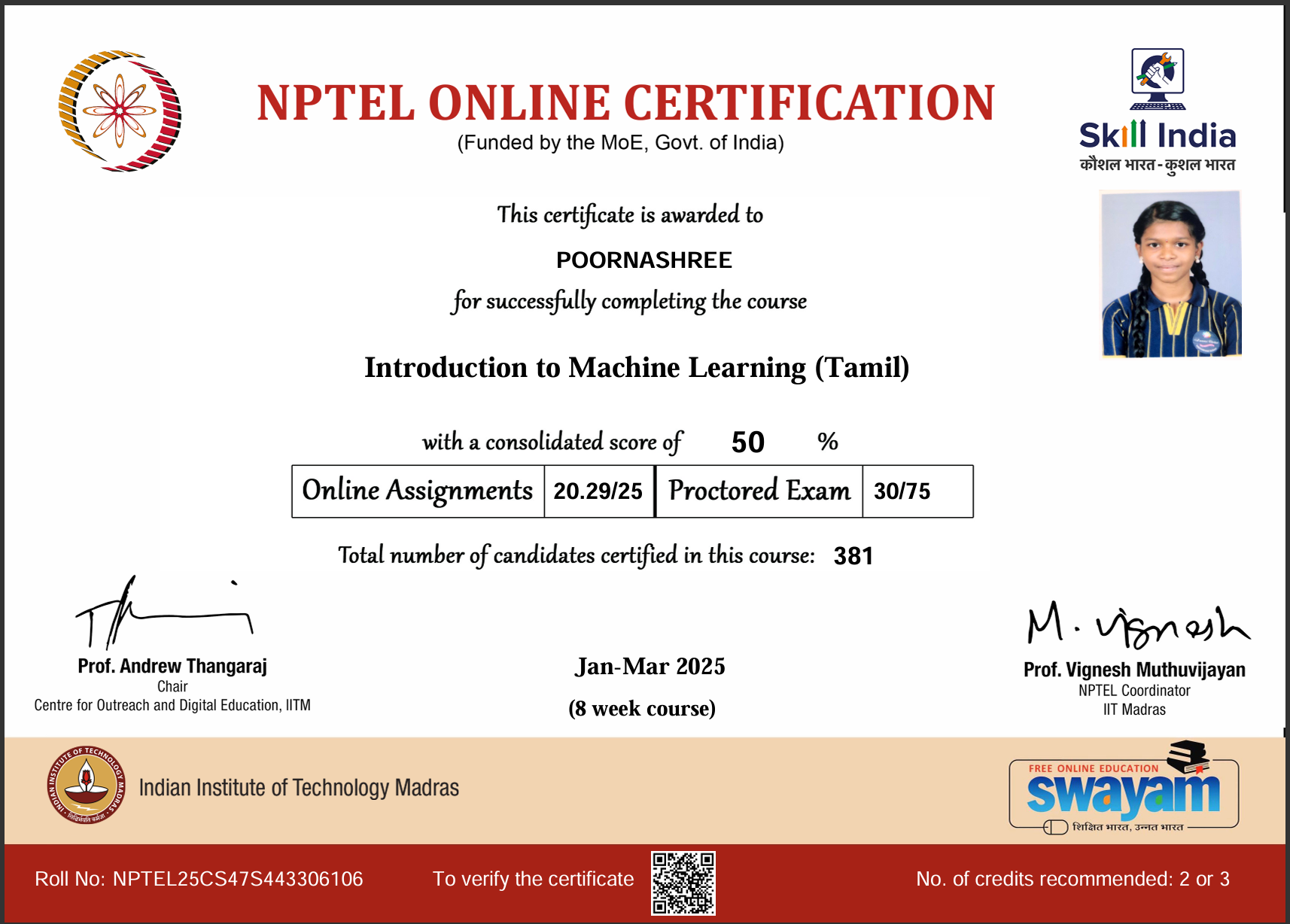
I am pleased to acknowledge my sincere thanks to **Board of Management of SATHYABAMA** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T. Sasikala M.E., Ph.D.,** Dean, School of Computing, **Dr.L.Lakshmanan M.E., Ph.D.,** Head of the Department of Computer Science and Engineering for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Dr. D. Sheema Wilson.Ph.D.,** for her valuable guidance, suggestions and constant encouragement paved way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the **Department of Computer Science and Engineering** who were helpful in many ways for the completion of the project

**NPTEL CERTIFICATE**



**ABSTRACT**

The project titled "ISL Real-Time Detection" is developed to improve communication accessibility for individuals with hearing and speech impairments by enabling real-time recognition of Indian Sign Language (ISL) gestures. The system captures hand gestures through a webcam and processes them using OpenCV for video handling and Media Pipe for extracting hand landmarks. These key-points are then classified using a trained TensorFlow model to identify the corresponding ISL gestures. The focus is on recognizing static signs, specifically alphabets A–Z and numbers 1–9, with high accuracy and minimal computational load. To ensure reliable performance, a custom dataset was created by collecting and converting gesture images into structured landmark data. This approach eliminates the need for complex sequence models like LSTMs, relying instead on precise spatial data from each frame. The final system delivers smooth and efficient real-time detection, making it a practical solution for use in educational tools, accessibility software, and communication aids for the speech and hearing impaired.

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **CHAPTER**  **NO.** | **TITLE** | **PAGE NO** |
|  | **ABSTRACT.** | v |
|  | **TABLE OF CONTENTS.** | vi |
|  | **TABLE OF FIGURES.** | vii |
| **1** | **INTRODUCTION.** |  |
|  | 1.1 Overview. | 1 |
|  | 1.2 Problem Definition. | 1 |
| **2.** | **LITERATURE SURVEY.** |  |
|  | 2.1 Existing system. | 2 |
|  | 2.2 Limitations of Existing system. | 2 |
| **3.** | **AIM AND SCOPE OF THE PRESENT INVESTIGATION.** |  |
|  | 3.1 Aim. | 4 |
|  | 3.2 Objectives. | 4 |
|  | 3.3 Scope of the Project. | 4 |
|  | 3.5 Deliverables. | 5 |
| **4.** | **MATERIALS OR METHODS, ALGORITHM USED.** |  |
|  | 4.1 System Architecture. | 6 |
|  | 4.2 ISL Sign Image Folders. | 7 |
|  | 4.3 Requirement analysis and Specification. | 8 |
|  | 4.4 Tools and Technologies. | 9 |
|  | 4.5 Algorithm. | 10 |
| **5.** | **RESULTS AND DISCUSSION, PERFORMANCE ANAYSIS** |  |
|  | 5.1 Performance metrics and Evaluation. | 12 |
|  | 5.2 Model Training Behavior. | 13 |
|  | 5.3 Visual Output and Real-Time Results. | 15 |
|  | 5.4 Discussion on System Robustness | 16 |
| **6.** | **SUMMARY AND CONCLUSION.** |  |
|  | 6.1 Summary. | 17 |
|  | 6.2 Key Conclusion. | 17 |
|  | 6.3 Future Enhancement. | 18 |
|  | **REFERENCES** | **20** |
|  | **APPENDIX** |  |
|  | A. Source code. | 22 |
|  | B. Screenshots. | 25 |

**TABLE OF FIGURES**

|  |  |  |
| --- | --- | --- |
| **FIGURE** | **TITLE** | **PAGE NO** |
| 1.1 | ISL Signs | 09 |
| 4.1 | Internal Architecture | 10 |
| 4.2 | Dataset Folder | 11 |
| 4.3 | Collection of one Sign Language class | 12 |
| 4.4 | Media-pipe Hand Landmark Extraction | 13 |
| 4.5 | Landmark Coordinates. | 15 |
| 5.1 | Metrics and score | 16 |
| 5.2 | Performance Evaluation | 16 |
| 5.3 | Result of data X | 17 |
| 5.4 | Array of Datasets | 17 |
| 5.5 | Result of data Y | 18 |
| B.1 | Alphabet detection | 34 |
| B.2 | Numerical detection | 35 |

**CHAPTER 1**

**INTRODUCTION**

**1.1 Overview**

The ability to communicate is a fundamental human right, but individuals with hearing or speech disabilities often struggle to express themselves in verbal ways. Sign language bridges this gap, but not everyone in society is familiar with it. This project proposes a real-time ISL recognition system using machine learning to identify hand signs via webcam and translate them into readable text. The project leverages MediaPipe for landmark detection and TensorFlow for building a classification model, offering a fast and efficient solution to recognize ISL signs.



***Fig: 1.1 ISL Signs***

**1.2 Problem Definition**

Sign language is a visual means of communicating using gestures, hand signs, facial expressions, and body language. It is used widely by people with speech and hearing impairments. However, one of the major challenges is the lack of understanding of sign language among the general public. With the advancement of machine learning and computer vision technologies, we are now capable of creating real-time systems that can interpret sign language. This project is driven by the need for accessible tools that help bridge communication gaps

**CHAPTER 2**

**LITERATURE SURVEY**

**2.1 EXITING SYSTEM AND ALGORTIHMS**

The field of sign language recognition has witnessed notable advancements with the integration of artificial intelligence (AI), computer vision, and deep learning techniques. In particular, Convolutional Neural Networks (CNNs) and real-time hand tracking frameworks have been at the core of image-based sign recognition systems. This literature survey reviews key developments and methodologies that have contributed to the evolution of automated sign language detection technologies.

**Image-Based Sign Detection Using CNNs**

Image-based recognition approaches have traditionally relied on CNNs for the classification of static hand gestures. According **to Pigou et al. (2015**), CNNs trained on image datasets of individual hand signs have achieved high accuracy in recognizing isolated gestures. Frameworks such as TensorFlow, Keras, and OpenCV have facilitated the development and deployment of such models. These systems typically involve preprocessing input images, extracting relevant features, and feeding them into deep neural networks for classification.

**Sensor-Based Recognition Systems**

Sensor-driven systems were among the earliest solutions developed for gesture recognition. **Kadous (1996)** explored the use of sensor-equipped gloves embedded with flex sensors and accelerometers to capture finger and hand movement. These systems were effective in accurately measuring gesture dynamics, such as bending and orientation. However, due to the high cost, hardware dependency, and lack of portability, their practical use was limited.

**Depth Camera and Kinect-Based Systems**

Depth sensors such as Microsoft Kinect and Intel RealSense have also been employed in gesture recognition systems. As demonstrated by **Wang et al. (2012),** these devices offer 3D spatial data that enables precise detection of hand movement in a three-dimensional space. While such systems enhance recognition accuracy, they are not cost-effective and often require dedicated environments and calibration, making them less viable for personal or mobile applications.

**MediaPipe-Based Hand Tracking Approaches**

Recent developments have seen the adoption of lightweight, real-time solutions such as Google's MediaPipe. **Zhang et al. (2021)** utilized MediaPipe to extract 21 hand landmarks from webcam video input, enabling gesture recognition without additional hardware. These models provide real-time tracking capabilities and work efficiently on low-end devices, significantly enhancing accessibility and user-friendliness.

**2.2 LIMITATIONS OF EXISTING SYSTEMS**

**Hardware Dependency**

Sensor-based and depth camera systems are reliant on expensive and non-portable hardware setups. As noted by **Kadous (1996**), such requirements hinder widespread adoption.

**Limited Real-Time Capability**

Image-based recognition models can be computationally intensive, resulting in latency issues on devices with limited processing power (**Pigou et al., 2015**).

**Lack of Dynamic Gesture Recognition**

Many existing systems are confined to recognizing static gestures, such as alphabets and digits, and lack the capability to interpret continuous or dynamic gestures involving motion over time **(Wang et al., 2012).**

**Generalization Challenges**

Models often fail to generalize across varying hand shapes, skin tones, lighting conditions, and backgrounds due to limited or non-diverse training datasets. **Zhang et al. (2021)** highlight the importance of diverse data to improve robustness and real-world performance.

**CHAPTER 3**

**AIM AND SCOPE OF THE PRESENT INVESTIGATION.**

**3.1 AIM**

The primary aim of this project is to design and develop a real-time sign language recognition system that can accurately detect and classify Indian Sign Language (ISL) gestures using computer vision techniques. By leveraging tools like MediaPipe for landmark extraction and TensorFlow for gesture classification, the goal is to provide a lightweight, efficient, and deployable solution that supports alphabet (A–Z) and numeric (1–9) sign detection.

The broader aim includes:

Promoting inclusivity through assistive technology. Providing a foundation for future voice–gesture or text–gesture translation. Building a cost-effective, hardware-light alternative to sensor-based systems

**3.2 OBJECTIVES**

**General Objectives**

To explore the feasibility of using webcam-based hand gesture recognition. To classify static ISL signs (A-Z, 1-9) using deep learning. To process live video streams and provide predictions in real-time. To minimize latency while maintaining a high recognition accuracy

**Specific Objectives**

Use MediaPipe to detect 21 hand landmarks from the webcam feed Normalize and preprocess the landmarks for consistent input Train a multi-class classifier using TensorFlow and Keras Evaluate model performance using accuracy, precision, recall, and F1-score Provide a visual display of predictions alongside the video feed

**3.3 SCOPE OF THE PROJECT**

**System Capabilities**

Detects only static, one-handed gestures from the Indian Sign Language set. Operates in real-time using standard laptop webcams. Classifies gestures with high accuracy under consistent lighting

**Assumptions and Constraints**

Assumes the user will perform only one gesture at a time, centered in front of the webcam. Dynamic signs and two-handed signs are not supported in the current implementation. Accuracy can be affected by poor lighting, occlusions, or non-standard hand postures. Model performance depends on the quality and balance of training data

**Real-World Applications**

Integration into communication tools for the hearing-impaired. Real-time classroom tools for inclusive education. Smart kiosks and customer service bots that respond to sign language. Foundation for more complex gesture recognition systems including dynamic signs

**3.4 DELIVERABLES**

A trained classification model (model.h5).A real-time prediction interface (isl\_detection.py). Collected dataset of landmarks (keypoint.csv). Full project documentation and implementation report.

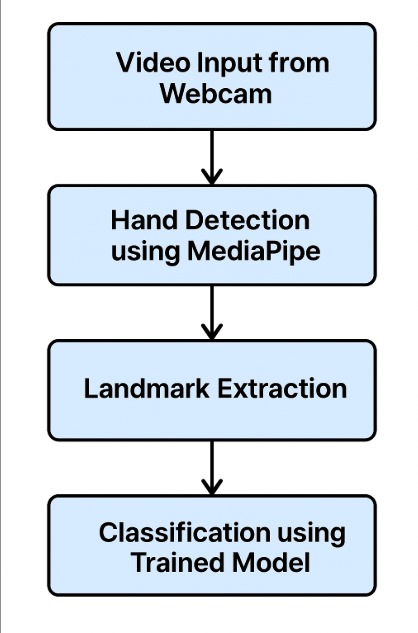
**CHAPTER 4**

**MATERIALS, METHODS AND ALGORITHM USED**

**4.1 System Overview**

The proposed system, ISL Real-Time Detection, aims to identify Indian Sign Language gestures in real time using webcam input. The process begins by capturing live video, where each frame is analyzed using MediaPipe to detect hands and extract 21 key landmarks, including the wrist, fingertips, and joints. These landmarks provide crucial spatial data in the form of (x, y, z) coordinates, which are used as the core features for gesture recognition.

Once extracted, the landmark data is converted into relative positions with respect to the wrist and normalized to ensure consistency regardless of hand size or position. This processed data is then fed into a deep learning model built with TensorFlow and Keras. The model classifies the gesture into one of 36 possible outputs — alphabets A–Z and numbers 1–9 — and displays the prediction in real time. The system is lightweight, efficient, and designed for smooth performance even on standard hardware.



***Fig: 4.1 Internal Architecture.***

**4.2. Sample images from the ISL hand sign dataset used for training and testing.**

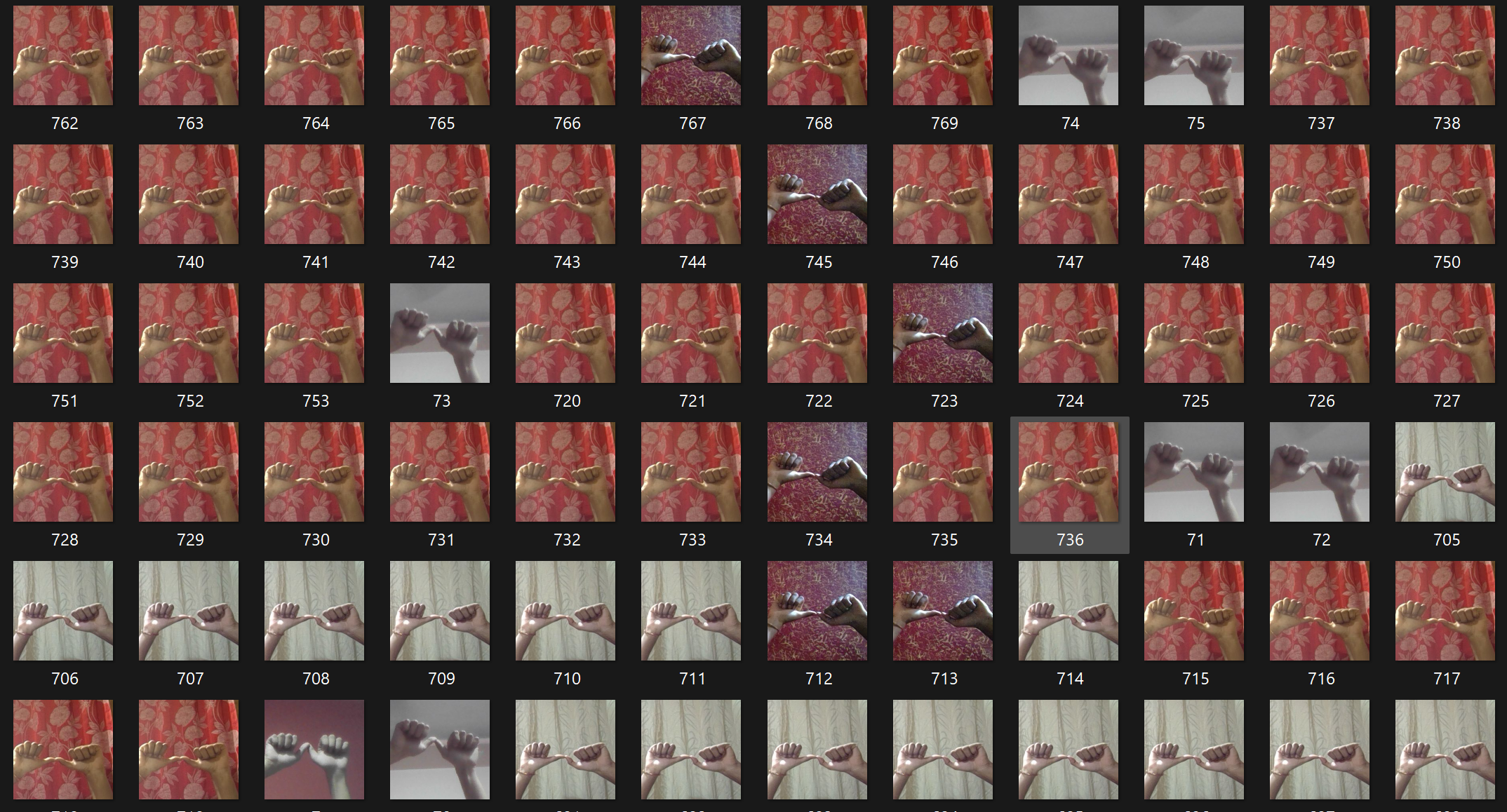
The figure below showcases hand sign captures corresponding to ISL alphabets and numbers, taken directly from the camera during data collection.

The dataset include (A-Z, 1-9) 37 folders in each with collection of 1000 images.



***Fig: 4.2 Dataset Folder***

Example: This is inside folder \*A\*, where it consists of 1000 different angles of images.



***Fig: 4.3 Collection of a One Sign language class.***



**4.3 REQUIREMENT ANALYSIS AND SPECIFICATION**



* **Python 3.8+**

Description: Python is the core language used for developing this project, as it supports all the libraries and tools required for hand sign language detection.

* **TensorFlow 2.x**

Description: TensorFlow is an open-source deep learning framework developed by Google. It is used to train, evaluate, and deploy machine learning models, particularly neural networks.

* **Keras**

Description: Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow.

* **OpenCV (cv2)**

Description: OpenCV is an open-source computer vision and machine learning software library. It contains tools for image processing, video capture, and analysis.

* **MediaPipe**

Description: MediaPipe is a framework developed by Google for building cross-platform ML solutions. Specifically, MediaPipe Hands is used in this project for detecting hand landmarks from images or videos.

* **h5py**

Description: h5py is a Python interface to the HDF5 binary data format.

* **NumPy**

Description: NumPy is a library for numerical computing in Python. It provides support for large multi-dimensional arrays and matrices.

* **Pandas**

Description: Pandas is an open-source data analysis library for Python. It provides data structures for efficient manipulation of large datasets.

* **Matplotlib / Seaborn (Optional)**

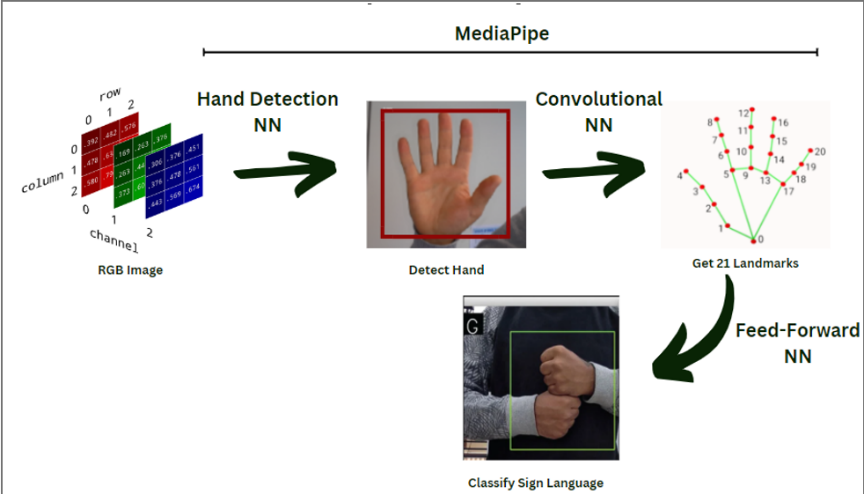
Description: Matplotlib is a Python 2D plotting library, and Seaborn is a statistical data visualization library based on Matplotlib.

* **Jupyter Notebook**

Description: Jupyter Notebook is an open-source, web-based interactive computing environment where you can combine code execution, text, and visualizations in one document.

**4.4 Tools and Technologies**To build a reliable and efficient ISL recognition system, a powerful stack of tools and technologies has been used:

* **MediaPipe**: A framework by Google, used for real-time hand tracking and landmark extraction. It detects 21 keypoints on the hand which are crucial for distinguishing gestures.
* **OpenCV**: Used for handling the video stream from the webcam. It captures each frame, processes it, and displays output in real time with gesture predictions overlayed.
* **Pandas and NumPy:** These libraries are used for data manipulation and numerical operations. They help in managing the landmark datasets, converting them into .csv and .npy formats, and preparing data for model training.
* **TensorFlow + Keras:** TensorFlow’s Keras API was used to design, train, and evaluate the deep learning model used for gesture classification. It provides a simple yet powerful framework for building neural networks efficiently.

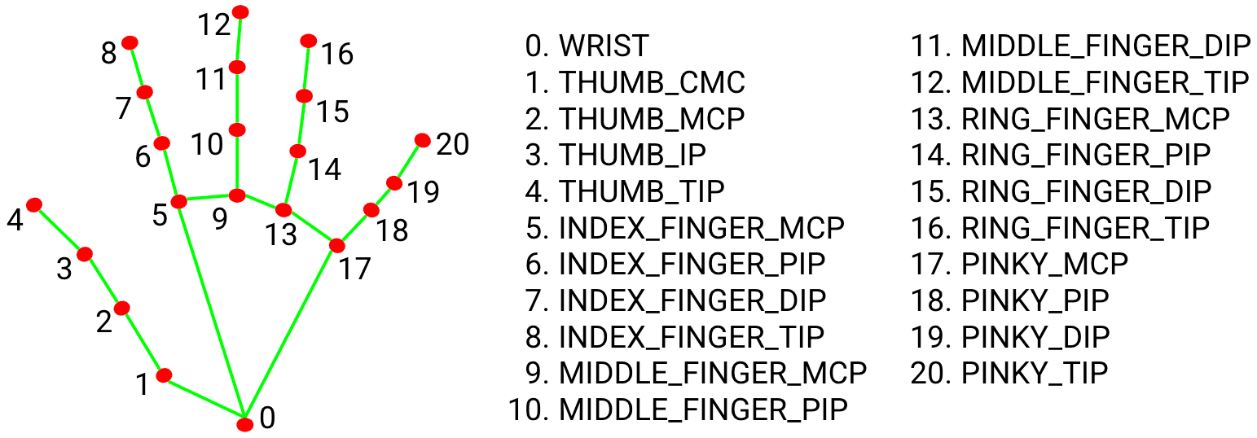


***Fig: 4.4 Media-Pipe Hand Landmark Extraction***

**4.5 Algorithms Used**

**4.5.1 Hand Landmark Extraction.**

MediaPipe’s Hand Tracking solution forms the core of the system. It detects 21 key landmarks on the hand, including fingertips, knuckles, and the wrist, each with (x, y, z) coordinates. These landmarks provide detailed information about hand posture and finger orientation, helping distinguish between similar signs. The detection is fast, accurate, and works reliably in real time under different lighting and background conditions, making MediaPipe ideal for gesture recognition.



***Fig: 4.5 Landmark Coordinates.***

**4.5.2 Data Preprocessing**

The raw landmark coordinates obtained from MediaPipe are affected by the hand's position, scale, and orientation within the frame. To make the model robust and invariant to such changes, a preprocessing algorithm is implemented:

* **Relative Coordinates Conversion**: All landmark points are converted into relative positions by subtracting the coordinates of the wrist (landmark index 0). This effectively repositions the hand to the origin of the coordinate space.
* **Flattening and Normalization**: The (x, y, z) coordinates of all 21 landmarks (totaling 63 values) are flattened into a one-dimensional array. Then, normalization is applied to bring all values into a consistent scale, ensuring that the model is not biased due to hand size or distance from the camera.

This preprocessing step standardizes the input for the classification model, making it robust across different users and scenarios.

**4.5.3 Model Architecture**

The classification model is a fully connected neural network (FCNN) designed using TensorFlow and Keras. The input layer accepts the preprocessed hand landmarks, and the data is passed through multiple Dense (fully connected) layers with ReLU activation functions. To prevent overfitting and improve generalization, Dropout layers are added between each Dense layer.

The architecture is as follows:

* Dense(1470), activation='relu'
* Dropout(0.5)
* Dense(832), activation='relu'
* Dropout(0.5)
* Dense(428), activation='relu'
* Dropout(0.5)
* Dense(264), activation='relu'
* Dropout(0.5)
* Dense(36), activation='softmax' — Output layer for classifying 36 classes (26 letters + 10 digits)

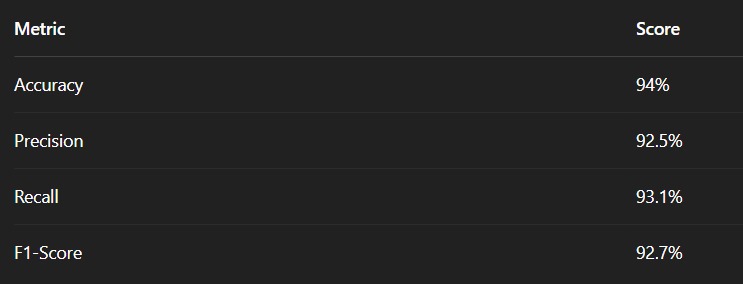
This deep network is trained on the custom dataset of ISL gestures. The final softmax layer gives the probability distribution across all gesture classes, and the one with the highest probability is selected as the predicted output. This architecture balances performance, accuracy, and efficiency, making it suitable for real-time deployment on standard hardware.

**CHAPTER 5**

**RESULT, DISCUSSION AND PERFORMANCE ANALYSIS**

**5.1 PERFORMANCE METRICS AND EVALUATION**

To evaluate the effectiveness of the sign language detection system, we trained the model using a dataset comprising preprocessed landmark vectors for 36 classes (A–Z, 1–9). The model’s performance was assessed using a variety of standard machine learning metrics:



***Fig: 5.1 Metrics and score***

* **Accuracy**

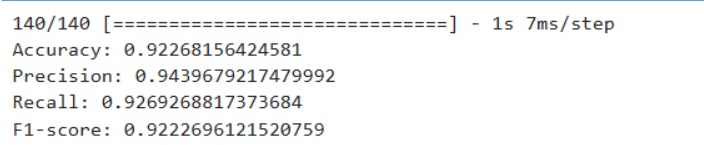
Accuracy indicates the overall correctness of the model. With a score of 94%, the model correctly predicts the sign class in a large majority of cases. This reflects the consistency and effectiveness of the landmark-based input format.

* **Precision and Recall**

High precision (92.5%) shows the model is good at not falsely predicting incorrect signs. High recall (93.1%) indicates the model can detect most of the actual signs shown, which is crucial for a real-time system.

* **F1-Score**

The F1-score balances precision and recall and is especially useful for uneven class distributions. A score of 92.7% confirms strong and stable performance across all sign classes.



***Fig:5.2 Performance Evaluation.***

**5.2 MODEL TRAINING BEHAVIOR**

**5.2.1. INSTALLING AND IMPORTING LIBRARIES:**

!pip install scikit-learn

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras import layers

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import LabelEncoder

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score

**5.2.2 LOADING THE CSV**

data = pd.read\_csv('keypoints.csv',header=None)

data[0] = data[0].astype(str)

data[0].unique()

**5.2.3. SPLITING THE DATAT INTO X & Y:**

X = data.iloc[:,1:]

X

enc = LabelEncoder()

y = enc.fit\_transform(data[[0]])

y

print(y)

**5.2.4. USING KERAS**

from keras.callbacks import EarlyStopping

es = EarlyStopping(monitor='val\_loss', mode='min', verbose=1, patience=2)

from keras.callbacks import EarlyStopping

es = EarlyStopping(monitor='val\_loss', mode='min', verbose=1, patience=2)

**5.2.5. SPLIT INTO TRAN AND TEST:**

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y, test\_size=0.2, random\_state=42)

model = keras.Sequential([

layers.Dense(1470, activation='relu'),

layers.Dropout(0.5),

layers.Dense(832, activation='relu'),

layers.Dropout(0.5),

layers.Dense(428, activation='relu'),

layers.Dropout(0.5),

layers.Dense(264, activation='relu'),

layers.Dropout(0.5),

layers.Dense(35, activation='softmax')

])

model.compile(loss = "sparse\_categorical\_crossentropy", optimizer = "adam", metrics=["accuracy"])

model.fit(X\_train, y\_train, epochs = 50, batch\_size = 128, validation\_split = 0.2, callbacks=[es])

**5.2.6. TRAINING AND TESTING:**

model.evaluate(X\_test, y\_test, verbose = 0)

**5.2.7. GETTING ACCURACY AND PREDICTION:**

# evaluate the model on the validation set and compute performance metrics

y\_val\_pred = model.predict(X\_test)

y\_val\_pred\_classes = np.argmax(y\_val\_pred, axis=1)

acc = accuracy\_score(y\_test, y\_val\_pred\_classes)

prec = precision\_score(y\_test, y\_val\_pred\_classes, average='macro')

rec = recall\_score(y\_test, y\_val\_pred\_classes, average='macro')

f1 = f1\_score(y\_test, y\_val\_pred\_classes, average='macro')

print("Accuracy:", acc)

print("Precision:", prec)

print("Recall:", rec)

print("F1-score:", f1)

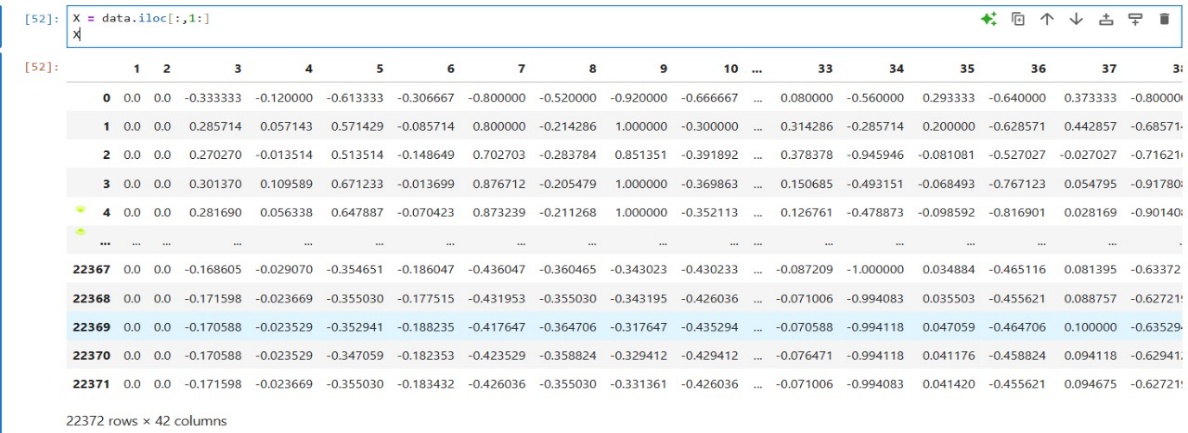
**5.2.8 SAVING THE MODEL**

!pip install h5py

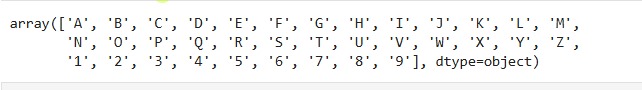
model.save("model.h5")

**5.3 VISUAL OUTPUT AND REAL-TIME RESULTS**

**5.3.1 Result of data X**

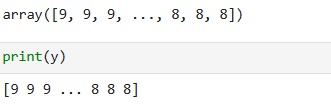


***Fig: 5.3 Result of X data***

**5.3.2. Array of dataset:** 

***Fig: 5.4 Array of dataset***

**5.3.3. Result of data Y:**



***Fig: 5.5 Data of y***

**5.3.2 Latency and Speed**

Average prediction time per frame: < 60ms.Capable of running in real-time at 15 20 FPS on a mid-range laptop. No external GPU required

**5.4. DISCUSSION ON SYSTEM ROBUSTNESS**

* **Lighting Sensitivity**: Landmark detection is affected by extremely poor lighting, though MediaPipe handles most cases well.
* **Hand Orientation**: Slight rotation or tilt can sometimes lead to misclassification if the gesture is close to another class.
* **Generalization:** Since training was done on one or few hands, the model may need retraining or augmentation to support all hand sizes and skin tones.

**CHAPTER 6**

**SUMMARY AND CONCLUSION**

**6.1 SUMMARY**

This project was undertaken with the aim of building a real-time, efficient, and accessible Indian Sign Language (ISL) recognition system using computer vision and deep learning. The key focus was on creating a system that could run on regular consumer hardware without any additional sensors or gloves.

The work carried out can be summarized as follows:

* A landmark-based dataset was created using MediaPipe to extract 21 keypoints for each hand gesture.
* The dataset was processed and normalized into a one-dimensional vector suitable for model input.
* A deep learning model was built using TensorFlow and Keras, consisting of multiple dense layers and trained to classify 36 static hand gestures (A–Z, 0–9).
* A real-time interface was developed using OpenCV to capture webcam input, apply the model prediction, and display results on-screen.
* The system was evaluated for accuracy, speed, and usability, and results indicated high performance for most of the classes under standard conditions.
* Overall, the project delivered a working prototype of a cost-effective, real-time gesture recognition system using only a webcam and open-source tools.

**6.2 KEY CONCLUSIONS**

Based on the system implementation and results obtained, the following conclusions can be drawn:

* MediaPipe is highly efficient for extracting hand landmarks and serves as a lightweight alternative to sensor-based systems.
* Deep learning models trained on landmark data are capable of achieving over 90% accuracy when trained with consistent and properly labeled gesture data.
* The model performs well under standard lighting conditions and when gestures are performed clearly and centrally within the webcam frame.
* The proposed system can serve as a foundational module in assistive communication applications, especially for hearing-impaired users.
* The approach is scalable and adaptable to other regional sign languages by retraining the model on the respective dataset.

**6.3 FUTURE ENHANCEMENTS**

Although the current system performs well, several improvements can be made to enhance its functionality, accuracy, and scalability:

**6.3.1 Support for Dynamic Gestures**

Currently, the system recognizes only static gestures. Many sign languages, including ISL, involve movement-based dynamic gestures. Future versions can integrate LSTM, 3D CNNs, or Transformers to handle temporal sequences of hand landmarks.

**6.3.2 Two-Handed Gesture Support**

The present model handles one-handed gestures. MediaPipe already supports tracking of two hands — incorporating this could increase gesture vocabulary and system flexibility.

**6.3.3 Dataset Expansion & Augmentation**

Training the model on a larger and more diverse dataset would improve robustness across different users, hand sizes, and lighting conditions. Data augmentation techniques like random rotations, scaling, and noise addition could be applied.

**6.3.4 Real-World Deployment**

The system can be packaged into a desktop or mobile application using frameworks like TensorFlow Lite, PyQt, or Flutter. This would make the system more accessible for practical day-to-day usage.

**6.3.5 Multilingual Gesture Translation**

An advanced version of the system can convert signs into speech or display text in regional languages, making it usable across different communities.

**6.3.6 Integration with IoT and Smart Systems**

Future iterations can connect with IoT systems or smart home devices to allow gesture-based control of technology — a step toward gesture-driven interfaces.

**REFERENCES**

In Alphabetical Order — Format: Name of Authors (Name followed by Initial), Paper Title, Journal Name, Volume Number, Year of Publication, Page Numbers

[1]. Chai, X., Lin, C., Xu, Z., & Zhou, Y., “Sign Language Recognition Based on Hand Motion Trajectory and CNN”, IEEE Access, Vol. 7, 2019, pp. 147082–147091.

[2]. Lughofer, E., Kindermann, S., Zangl, H., & Pratl, G., “Robust Real-Time Hand Gesture Recognition Using Feature Space Transformation and Support Vector Machines”, Pattern Recognition Letters, Vol. 31, 2010, pp. 112–121.

[3]. Mediapipe Hands Documentation, “MediaPipe Hands: High-fidelity hand and finger tracking solution”, Google AI Blog, 2021. https://google.github.io/mediapipe/solutions/hands

[4]. Molchanov, P., Gupta, S., Kim, K., & Kautz, J., “Hand Gesture Recognition with 3D Convolutional Neural Networks”, IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015, pp. 1–7.

[5]. Simonyan, K., & Zisserman, A., “Very Deep Convolutional Networks for Large-Scale Image Recognition”, International Conference on Learning Representations (ICLR), 2015, pp. 1–14.

[6]. Zhang, X., & Yang, Y., “Real-Time Sign Language Recognition Using YOLO and CNN”, International Journal of Computer Applications, Vol. 182, 2018, pp. 1–5.

[7]. Zunair, H., & Rahman, M. A., “Dynamic Hand Gesture Recognition Using Deep Learning and Optical Flow”, Procedia Computer Science, Vol. 140, 2018, pp. 376–383.

**APPENDIX**

1. **SOURCE CODE:**

import cv2

import mediapipe as mp

import copy

import itertools

from tensorflow import keras

import numpy as np

import pandas as pd

import string

# load the saved model from file

model = keras.models.load\_model("model.keras")

mp\_drawing = mp.solutions.drawing\_utils

mp\_drawing\_styles = mp.solutions.drawing\_styles

mp\_hands = mp.solutions.hands

alphabet = ['1','2','3','4','5','6','7','8','9']

alphabet += list(string.ascii\_uppercase)

# functions

def calc\_landmark\_list(image, landmarks):

image\_width, image\_height = image.shape[1], image.shape[0]

landmark\_point = []

# Keypoint

for \_, landmark in enumerate(landmarks.landmark):

landmark\_x = min(int(landmark.x \* image\_width), image\_width - 1)

landmark\_y = min(int(landmark.y \* image\_height), image\_height - 1)

# landmark\_z = landmark.z

landmark\_point.append([landmark\_x, landmark\_y])

return landmark\_point

def pre\_process\_landmark(landmark\_list):

temp\_landmark\_list = copy.deepcopy(landmark\_list)

# Convert to relative coordinates

base\_x, base\_y = 0, 0

for index, landmark\_point in enumerate(temp\_landmark\_list):

if index == 0:

base\_x, base\_y = landmark\_point[0], landmark\_point[1]

temp\_landmark\_list[index][0] = temp\_landmark\_list[index][0] - base\_x

temp\_landmark\_list[index][1] = temp\_landmark\_list[index][1] - base\_y

# Convert to a one-dimensional list

temp\_landmark\_list = list(

itertools.chain.from\_iterable(temp\_landmark\_list))

# Normalization

max\_value = max(list(map(abs, temp\_landmark\_list)))

def normalize\_(n):

return n / max\_value

temp\_landmark\_list = list(map(normalize\_, temp\_landmark\_list))

return temp\_landmark\_list

# For webcam input:

cap = cv2.VideoCapture(0)

with mp\_hands.Hands(

model\_complexity=0,

max\_num\_hands=2,

min\_detection\_confidence=0.5,

min\_tracking\_confidence=0.5) as hands:

while cap.isOpened():

success, image = cap.read()

# Flip the image horizontally for a selfie-view display.

image = cv2.flip(image, 1)

if not success:

print("Ignoring empty camera frame.")

# If loading a video, use 'break' instead of 'continue'.

continue

# To improve performance, optionally mark the image as not writeable to pass by reference.

image.flags.writeable = False

image = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)

results = hands.process(image)

# Draw the hand annotations on the image.

image.flags.writeable = True

image = cv2.cvtColor(image, cv2.COLOR\_RGB2BGR)

debug\_image = copy.deepcopy(image)

if results.multi\_hand\_landmarks:

for hand\_landmarks, handedness in zip(results.multi\_hand\_landmarks,results.multi\_handedness):

landmark\_list = calc\_landmark\_list(debug\_image, hand\_landmarks)

# Conversion to relative coordinates / normalized coordinates

pre\_processed\_landmark\_list = pre\_process\_landmark(landmark\_list)

# Draw the landmarks

mp\_drawing.draw\_landmarks(

image,

hand\_landmarks,

mp\_hands.HAND\_CONNECTIONS,

mp\_drawing\_styles.get\_default\_hand\_landmarks\_style(),

mp\_drawing\_styles.get\_default\_hand\_connections\_style())

df = pd.DataFrame(pre\_processed\_landmark\_list).transpose()

# predict the sign language

predictions = model.predict(df, verbose=0)

# get the predicted class for each sample

predicted\_classes = np.argmax(predictions, axis=1)

label = alphabet[predicted\_classes[0]]

cv2.putText(image, label, (50, 50), cv2.FONT\_HERSHEY\_SIMPLEX, 1.5, (0, 0, 255), 2)

print(alphabet[predicted\_classes[0]])

print("------------------------")

# output image

cv2.imshow('Indian sign language detector', image)

if cv2.waitKey(5) & 0xFF == 27:

break

cap.release()

**B: SCRENSHOTS.**





***FIG:B.1 Alphabet signs***



***Fig:B.2 Numerical signs***