***Sign Language Detection Using Computer Vision***

*By Rimjhim Singh ,Sudarshan Goyal*

**Introduction**

Sign language detection plays a crucial role in enhancing communication for the hearing-impaired community. With advancements in computer vision and machine learning, there is a growing interest in automating the recognition of sign language gestures using various techniques. This report outlines a system designed to detect and classify American Sign Language (ASL) gestures through the integration of computer vision libraries and machine learning models, specifically utilizing MediaPipe and OpenCV.

**System Overview**

The presented code implements a sign language detection system that processes live video input from a webcam, identifies hand landmarks using MediaPipe, and classifies the gestures based on pre-trained machine learning models. The following components constitute the core of this system:

1. **Data Collection and Model Training**:
   * Before the execution of the detection code, a machine learning model is trained on hand landmark data corresponding to different sign language gestures. The model is then saved as a pickle file (model.p), which is loaded during the execution of the code.
2. **Hand Landmark Detection**:
   * The MediaPipe library provides a robust solution for real-time hand tracking and landmark detection. It detects and represents the hand's key points (landmarks) through a 3D coordinate system.
3. **Gesture Recognition**:
   * The collected landmark data is fed into the trained model for prediction. The system maps the predicted output to corresponding sign language letters using a predefined label dictionary.

**Code Implementation**

The following is an explanation of the key sections of the provided code, which is organized into distinct functional blocks.

**1. Importing Libraries and Loading the Model**

import pickle

import cv2

import mediapipe as mp

import numpy as np

# Load the model from the pickle file

model\_dict = pickle.load(open('./model.p', 'rb'))

model = model\_dict['model']

This section imports necessary libraries: pickle for model loading, cv2 for video capture and image processing, mediapipe for hand detection, and numpy for numerical operations. The model, which is stored in a pickle file, is loaded into memory.

**2. Defining Labels and Initializing Video Capture**

# Define the labels manually (if you know them)

labels\_dict = {0: 'A', 1: 'B', 2: 'C', 3: 'D', 4: 'E', 5: 'F'} # Modify based on your label set

cap = cv2.VideoCapture(0)

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=False, min\_detection\_confidence=0.3)

In this section, a dictionary maps integer labels to corresponding sign language characters (A-F in this case). The code initializes the webcam using cv2.VideoCapture(0), and sets up the MediaPipe hands module for real-time hand detection with a minimum detection confidence threshold of 0.3.

**3. Main Loop for Real-Time Gesture Detection**

while True:

data\_aux = []

ret, frame = cap.read()

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,

hand\_landmarks,

mp\_hands.HAND\_CONNECTIONS,

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

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

)

# Collect (x, y) coordinates of all landmarks

for i in range(len(hand\_landmarks.landmark)):

x = hand\_landmarks.landmark[i].x

y = hand\_landmarks.landmark[i].y

data\_aux.append(x)

data\_aux.append(y)

# Ensure the data has the correct feature length (42 for one hand, 84 for two)

if len(data\_aux) == 42:

prediction = model.predict([np.asarray(data\_aux)])

# Use the manually defined labels\_dict to get the predicted character

predicted\_character = labels\_dict[int(prediction[0])]

# Display the prediction on the frame

cv2.putText(frame, predicted\_character, (30, 100), cv2.FONT\_HERSHEY\_SIMPLEX, 1.3, (0, 0, 0), 3, cv2.LINE\_AA)

cv2.imshow('frame', frame)

if cv2.waitKey(1) & 0xFF == ord('q'):

break

This section contains the main loop that continuously captures video frames from the webcam. The following steps are performed in each iteration:

* The frame is read and converted to RGB format for processing.
* The MediaPipe hands model detects any hands present in the frame. If hands are detected, their landmarks are drawn on the frame for visualization.
* The coordinates of the detected landmarks are collected into a list, data\_aux. This list contains the normalized (x, y) coordinates of 21 hand landmarks.
* If 21 landmarks are detected (resulting in 42 features), these features are passed to the loaded model for prediction. The predicted label is then converted to the corresponding sign language character using the labels\_dict.
* Finally, the predicted character is displayed on the video frame, and the updated frame is shown in a window titled "frame".

1. **Clean Up**

cap.release()

cv2.destroyAllWindows()

Upon exiting the loop (when the 'q' key is pressed), the video capture is released, and all OpenCV windows are closed to clean up resources.

**Conclusion**

The code provided demonstrates a fundamental approach to real-time sign language detection using computer vision techniques. By leveraging the capabilities of MediaPipe for hand landmark detection and a pre-trained machine learning model for gesture classification, the system effectively identifies hand signs from live video input. This framework can be expanded further to recognize more complex gestures, improve accuracy, and incorporate additional features such as context-aware recognition or integration with natural language processing for more effective communication tools.

Such advancements could lead to improved accessibility and communication for the hearing-impaired community, showcasing the potential of combining machine learning and computer vision technologies in everyday applications.