Machine Learning Project

Documentation

**Topic:**

To develop a system that converts Indian Sign Language (ISL) into plain English, using either video or photo inputs, and outputs text or audio translations.

Table of Contents

[Abstract 3](#_Toc174232612)

[Introduction 4](#_Toc174232613)

[*Existing system* 8](#_Toc174232614)

[*Proposed system* 11](#_Toc174232615)

[*output* 18](#_Toc174232616)

[*conclusion* 19](#_Toc174232617)

[references 21](#_Toc174232618)

# Abstract

Hand sign detection belongs to the rapidly growing areas of computer vision and machine learning. Translation of hand movements into digital signals opens completely new perspectives for human-machine interaction, translation of sign language, and even augmented reality. In this paper, methods and technologies involved in hand sign detection are overviewed: data acquisition, preprocessing, feature extraction, and classification techniques. This overview represents current approaches known, among others, as Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), and indicates the challenges that need to be overcome for accurate and real-time detection, including hand shape and size variations, as well as environmental conditions. Moreover, this documentation also covers the integration of hand sign detection into practical applications and provides an outlook on future research directions. The authors foresee this document to be quite a comprehensive reference with regard to synthesizing recent advancements and giving practical implementation guidelines concerning hand sign detection technology to researchers and practitioners across the globe looking to exploit this technology.

# Introduction

Hand sign detection is at the intersection of computer vision, machine learning, and human–computer interaction; it forms one of the most critical technologies in translating hand gestures into related digital commands. The area has attracted many braces due to the possibility of applying it in such diversified fields as communication for the deaf and dumb, gesture-based user interfaces, virtual and augmented reality-based experiences, and robotics.

Basically, detection of hand signs involves visual identification and interpretation of some specified hand gestures or signs in an image or video stream. It is, therefore, a critical step towards developing systems that might establish communication with users in a more intuitive and natural way by closing the gap between human actions and digital responses. It will facilitate accessibility, provide new interaction methods, and further allow the smooth integration of gesture controls across a wide range of contexts in which technology is used.

In most cases, while developing the hand sign detection system, a number of stages are normally followed, which are:

1.**Data Acquisition:** Obtaining a varied dataset of hand gestures is the first step. This may be obtained from devices such as cameras, depth sensors, or wearable sensors. To train robust and efficient models for detection, high-quality and representative data are desired at the beginning.

2. **Preprocessing:**This step is often necessary to have uniformity in data and for good model performance. It typically includes normalization, noise reduction, and extraction of hand regions so that the model focuses on the important parts of the input.

3. **Feature Extraction:**Here, efficient features have to be extracted so that it becomes possible to distinguish between various hand signs. This hence entails identifying and isolating key characteristics of hand gestures, which in this case involve shape, position, and patterns of movement.

4. **Classification:** The extracted features are then used in classifying the hand signs against predefined categories. At this stage, the machine learning algorithms, notably CNN and RNN, play important roles in learning to recognize and differentiating between various gestures based on the features.

5. **Integration and Application:** After the development and validation of a hand-sign detection model, it needs to be integrated with applications. This step involves the incorporation of the model into software or hardware systems so that it can run real-time data processing for gesture-based interactions.

Thus, a number of challenges still exist in hand sign detection. Variability in hand shapes, sizes, and colors, and differences in light conditions and background, may affect the accuracy of the detection. Further, how to realize real-time performance with robustness toward diverse gestures and movements is part of future work.

The documentation brings forth all the details entailing hand sign detection, together with methodologies and technologies that have been associated with different challenges in this area. Consequently, the study will turn out very helpful for researchers, developers, and practitioners who would like to unlock the power of hand sign detection for many innovative applications and enriched human-computer interaction. **

Indian sign language

Indian Sign Language (ISL) is a rich and expressive visual language used by the Deaf and hard-of-hearing communities in India. It represents a crucial means of communication, cultural expression, and social integration for individuals who are deaf or have hearing impairments. ISL, distinct from other sign languages used around the world, has evolved uniquely within the Indian cultural and linguistic context, incorporating a diverse array of signs and gestures reflective of the country's multilingual and multicultural heritage.

Understanding and documenting Indian Sign Language is vital for several reasons. First, it facilitates greater inclusivity by improving communication access for the Deaf community, fostering educational opportunities, and enabling more effective engagement in various social and professional settings. Second, it plays a significant role in preserving and promoting the cultural identity of Deaf individuals, ensuring that their language and traditions are recognized and valued.

This documentation aims to provide a comprehensive overview of Indian Sign Language, addressing its key aspects including:

1. **Historical and Cultural Context:** An exploration of the origins and development of ISL, its relationship with other sign languages, and its role within the broader Indian cultural landscape.

2. **Linguistic Structure:** An examination of the linguistic features of ISL, including its phonology, syntax, and semantics. This section delves into the structure of ISL signs, the use of space and movement, and how these elements combine to form meaningful communication.

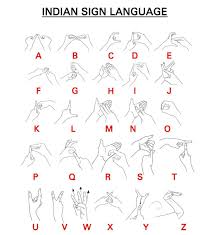
3. **Common Signs and Phrases:** A practical guide to commonly used ISL signs and phrases, providing visual representations and descriptions to aid in learning and application. This includes everyday communication, educational terms, and emergency signs.

4. **Educational Resources and Training:** An overview of resources available for learning ISL, including formal educational programs, online tools, and community-based initiatives. This section highlights opportunities for acquiring proficiency in ISL and encourages further study and practice.

5. **Technological Integration:** Insights into how technology can support and enhance ISL usage, including tools for sign language recognition, translation applications, and digital resources that facilitate communication between Deaf and hearing individuals.

6. **Challenges and Future Directions:** A discussion of the challenges faced by the Deaf community in India regarding the recognition and standardization of ISL, and the ongoing efforts to address these issues. This section also explores potential future developments in the field, including research opportunities and technological advancements.

By offering a detailed examination of Indian Sign Language, this documentation aims to bridge gaps in understanding and promote greater engagement with ISL. It serves as a valuable resource for educators, researchers, policymakers, and anyone interested in supporting the Deaf community and fostering inclusive communication practices.

**

# Existing system

The field of hand sign detection has witnessed significant advancements with the development and deployment of various systems that leverage computer vision and machine learning technologies. These existing systems utilize different approaches and technologies to achieve accurate and efficient hand sign recognition. This section reviews several prominent systems and frameworks currently used for hand sign detection, highlighting their methodologies, strengths, and limitations.

1. Sign Language Recognition Systems

Microsoft Kinect: One of the pioneering systems for hand sign detection, the Microsoft Kinect utilizes depth sensors and RGB cameras to capture 3D data of users’ hand gestures. Kinect’s hand tracking capabilities are integrated into various applications, including sign language recognition. The system employs machine learning algorithms to classify gestures based on depth and visual features, though it has limitations in terms of accuracy with complex gestures and in varying lighting conditions.

Leap Motion: Leap Motion provides a high-precision hand tracking system using infrared cameras and LEDs. It enables detailed gesture recognition by capturing finger movements and hand positions with high accuracy. The Leap Motion Controller is often used in applications for virtual reality and interactive environments, offering robust performance but requiring specific hardware setup and calibration.

2. Convolutional Neural Networks (CNNs) for Hand Sign Detection

MediaPipe Hands: Developed by Google, MediaPipe Hands is a framework that utilizes a CNN-based model for real-time hand tracking and gesture recognition. It provides high accuracy in detecting hand landmarks and supports a variety of hand gestures. MediaPipe’s lightweight design and integration with mobile and web applications make it a versatile solution. However, its performance can be affected by occlusions and complex background scenarios.

OpenPose: OpenPose, developed by the Carnegie Mellon Perceptual Computing Lab, is an open-source library for multi-person detection and pose estimation, including hand gestures. It uses a deep learning approach to provide accurate hand and body pose estimation. While OpenPose offers comprehensive pose tracking, its computational demands can be high, necessitating powerful hardware for real-time applications.

3. Gesture Recognition via Recurrent Neural Networks (RNNs)

DeepSign: DeepSign is a system that leverages RNNs, specifically Long Short-Term Memory (LSTM) networks, to recognize sign language gestures from sequential video frames. This approach captures temporal dynamics and movement patterns of hand gestures, which enhances recognition accuracy for continuous gestures. DeepSign is effective for dynamic sign language interpretation but requires extensive training data and computational resources.

Sign Language Recognition using 3D CNNs: Some systems employ 3D Convolutional Neural Networks to analyze temporal information from video sequences. These models can process spatiotemporal features of hand gestures and are used for recognizing continuous sign language streams. While 3D CNNs provide high accuracy, they are computationally intensive and necessitate large-scale datasets for training.

4. Wearable Sensor-Based Systems

SignAloud Gloves: SignAloud is a wearable glove equipped with sensors that detect hand movements and finger positions. The glove translates physical gestures into digital signals, which are then processed by a machine learning model to identify specific signs. This system offers high accuracy and direct gesture-to-speech translation but requires users to wear specialized equipment and may have limitations in terms of comfort and usability.

Myo Armband: The Myo Armband uses electromyography (EMG) sensors to capture muscle activity in the forearm, which is then interpreted to recognize hand gestures. This approach allows for gesture recognition without requiring visual input, making it useful in various environments. However, the system’s accuracy can be affected by muscle fatigue and variations in user anatomy.

Summary

Existing systems for hand sign detection utilize a range of technologies, from depth sensors and high-precision tracking devices to advanced neural networks and wearable sensors. Each system has its strengths, such as high accuracy, real-time performance, and ease of integration, but also faces limitations related to computational demands, environmental conditions, and user requirements. Continued advancements in machine learning, computer vision, and sensor technologies are expected to address these challenges and improve the effectiveness of hand sign detection systems.

# Proposed system

Hardware requirements:

* Hard Disk : 500GB and above
* RAM : 4GB and above
* Processor : 13 and above

Software requirements:

* Operating system : Windows 7, 8 , 10 (64bit),11(64bit)
* Software : Python
* Tools : Visual Studio Code

**Source code:**

**Generating dataset:**

import cv2

import mediapipe as mp

import numpy as np

import os

mp\_hands = mp.solutions.hands

hands = mp\_hands.Hands()

mp\_drawing = mp.solutions.drawing\_utils

cap = cv2.VideoCapture(0)

frame\_count = 0

max\_images = 1000

output\_dir = 'C:/Users/HP/Desktop/whiskey/pre final/dataimage/stop'

while cap.isOpened() and frame\_count < max\_images:

    success, image = cap.read()

    if not success:

        print("Ignoring empty camera frame.")

        continue

    image = cv2.cvtColor(cv2.flip(image, 1), cv2.COLOR\_BGR2RGB)

    image.flags.writeable = False

    results = hands.process(image)

    image.flags.writeable = True

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

    blank\_image = np.zeros((image.shape[0], image.shape[1], 3), dtype=np.uint8)

    if results.multi\_hand\_landmarks:

        for hand\_landmarks in results.multi\_hand\_landmarks:

            mp\_drawing.draw\_landmarks(

                image, hand\_landmarks, mp\_hands.HAND\_CONNECTIONS)

            mp\_drawing.draw\_landmarks(

                blank\_image, hand\_landmarks, mp\_hands.HAND\_CONNECTIONS)

        cv2.imwrite(os.path.join(output\_dir, f'hand\_landmarks\_{frame\_count}.png'), blank\_image)

        frame\_count += 1

    cv2.imshow('MediaPipe Hands', image)

    if cv2.waitKey(5) & 0xFF == 27:

        break

cap.release()

cv2.destroyAllWindows()

**model:**

import os

import shutil

import random

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from keras.models import Sequential

from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,Dropout

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, BatchNormalization

from tensorflow.keras.callbacks import EarlyStopping

# path for the dataset

data\_path = r"C:\Users\HP\Desktop\final project\dataforprocessing\dataimage"

classes=['A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','skip','space','T','U','V','W','X','Y','Z']

# Paths for train, valid, and test sets

train\_path = os.path.join(data\_path, 'train')

valid\_path = os.path.join(data\_path, 'valid')

test\_path = os.path.join(data\_path, 'test')

# Creating directories for train, valid, and test sets

for path in [train\_path, valid\_path, test\_path]:

    if not os.path.exists(path):

        os.makedirs(path)

        for class\_name in classes:

            os.makedirs(os.path.join(path, class\_name), exist\_ok=True)

train\_split = 0.7

valid\_split = 0.2

test\_split = 0.1

def split\_data(class\_name):

    class\_dir = os.path.join(data\_path, class\_name)

    images = os.listdir(class\_dir)

    random.shuffle(images)

    train\_size = int(train\_split \* len(images))

    valid\_size = int(valid\_split \* len(images))

    train\_images = images[:train\_size]

    valid\_images = images[train\_size:train\_size + valid\_size]

    test\_images = images[train\_size + valid\_size:]

    return train\_images, valid\_images, test\_images

# Moving files into the test,valid,train folders

for class\_name in classes:

    train\_images, valid\_images, test\_images = split\_data(class\_name)

    for image in train\_images:

        shutil.move(os.path.join(data\_path, class\_name, image), os.path.join(train\_path, class\_name, image))

    for image in valid\_images:

        shutil.move(os.path.join(data\_path, class\_name, image), os.path.join(valid\_path, class\_name, image))

    for image in test\_images:

        shutil.move(os.path.join(data\_path, class\_name, image), os.path.join(test\_path, class\_name, image))

#normalization process

train\_datagen = ImageDataGenerator(rescale=1./255,rotation\_range=20,width\_shift\_range=0.2,height\_shift\_range=0.2,shear\_range=0.2,zoom\_range=0.2,horizontal\_flip=True,fill\_mode='nearest')

valid\_datagen = ImageDataGenerator(rescale=1./255)

test\_datagen = ImageDataGenerator(rescale=1./255)

# Creating Batches

train\_batches = train\_datagen.flow\_from\_directory(

    directory=train\_path, target\_size=(128, 128), classes=classes, batch\_size=32, class\_mode='categorical',color\_mode='grayscale'

)

valid\_batches = valid\_datagen.flow\_from\_directory(

    directory=valid\_path, target\_size=(128, 64), classes=classes, batch\_size=32, color\_mode='grayscale'

)

test\_batches = test\_datagen.flow\_from\_directory(

    directory=test\_path, target\_size=(128, 128), classes=classes, batch\_size=32, color\_mode='grayscale',shuffle=False

)

 #creating the cnn model

model = Sequential([

    Conv2D(32, (3, 3), activation='relu', input\_shape=(128, 128, 1)),

    BatchNormalization(),

    MaxPooling2D(2, 2),

    Dropout(0.2),

    Conv2D(64, (3, 3), activation='relu'),

    BatchNormalization(),

    MaxPooling2D(2, 2),

    Dropout(0.2),

    Conv2D(128, (3, 3), activation='relu'),

    BatchNormalization(),

    MaxPooling2D(2, 2),

    Dropout(0.2),

    Conv2D(256, (3, 3), activation='relu'),

    BatchNormalization(),

    MaxPooling2D(2, 2),

    Dropout(0.2),

    Flatten(),

    Dense(128, activation='relu'),

    BatchNormalization(),

    Dropout(0.2),

    Dense(28, activation='softmax')

])

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

early\_stopping = EarlyStopping(monitor='val\_loss', patience=7, restore\_best\_weights=True)

model.fit(train\_batches, epochs=30, validation\_data=valid\_batches,callbacks=[early\_stopping])

model.save('saved\_model.keras')

**load the model:**

import cv2

import mediapipe as mp

import numpy as np

from tensorflow.keras.models import load\_model

from tensorflow.keras.preprocessing.image import img\_to\_array

import os

from collections import Counter

from moviepy.video.io.VideoFileClip import VideoFileClip

from gtts import gTTS

from spellchecker import SpellChecker

def correct\_and\_convert\_text\_to\_speech(text, filename="output.mp3"):

    spell = SpellChecker()

    corrected\_text = []

    words = text.split()

    misspelled = spell.unknown(words)

    for word in words:

        correction = spell.correction(word) if word in misspelled else word

        corrected\_text.append(correction if correction else word)

    corrected\_text = ' '.join(corrected\_text)

    print(f"Corrected Text: {corrected\_text}")

    tts = gTTS(text=corrected\_text, lang='en')

    tts.save(filename)

def save\_preprocessed\_image(image, filename):

    folder = r'media'

    if not os.path.exists(folder):

        os.makedirs(folder)

    filepath = os.path.join(folder, filename)

    cv2.imwrite(filepath, image)

def split\_video(video\_path, clip\_duration=5, output\_dir='video'):

    video = VideoFileClip(video\_path)

    if not os.path.exists(output\_dir):

        os.makedirs(output\_dir)

    total\_duration = video.duration

    num\_clips = int(total\_duration // clip\_duration)

    '''if total\_duration % clip\_duration != 0:

        num\_clips += 1'''

    #splitting according to clip duration

    for i in range(num\_clips):

        start\_time = i \* clip\_duration

        end\_time = min((i + 1) \* clip\_duration, total\_duration)

        subclip = video.subclip(start\_time, end\_time)

        # saving each clip to video folder

        subclip\_filename = os.path.join(output\_dir, f"clip\_{i+1}.mp4")

        subclip.write\_videofile(subclip\_filename, codec="libx264")

model=load\_model(r"models/rizwinmodel3.keras")

frame\_count=0

class\_names=['a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','skip','space','t','u','v','w','x','y','z']

mp\_hands = mp.solutions.hands

hands = mp\_hands.Hands(static\_image\_mode=False, max\_num\_hands=2, min\_detection\_confidence=0.5)

mp\_drawing = mp.solutions.drawing\_utils

split\_video(r"pred.mp4")

# video folder

video\_folder = r"video"

result\_string = ""

def preprocess\_image(image, target\_size):

    image = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

    image = cv2.resize(image, target\_size)

    image = img\_to\_array(image)

    image = np.expand\_dims(image, axis=0)

    image = image.astype("float") / 255.0

    return image

for video\_file in os.listdir(video\_folder):

    if video\_file.lower().endswith(('.mp4', '.avi', '.mov', '.mkv')):

        video\_path = os.path.join(video\_folder, video\_file)

        print(f"Processing video: {video\_path}")

        cap = cv2.VideoCapture(video\_path)

        predictions = []

        while cap.isOpened():

            success, frame = cap.read()

            if not success or frame is None:

                break

            # BGR image to RGB

            rgb\_frame = cv2.cvtColor(cv2.flip(frame,1), cv2.COLOR\_BGR2RGB)

            # detect hand

            rgb\_frame.flags.writeable = False

            result = hands.process(rgb\_frame)

            rgb\_frame.flags.writeable = True

            # initialize the black\_image ONCE per frame

            black\_image = np.zeros((rgb\_frame.shape[0], rgb\_frame.shape[1], 3), dtype=np.uint8)

            if result.multi\_hand\_landmarks:

                for hand\_landmarks in result.multi\_hand\_landmarks:

                    frame\_copy = frame.copy()

                    mp\_drawing.draw\_landmarks(

                        black\_image,

                        hand\_landmarks,

                        mp\_hands.HAND\_CONNECTIONS,

                        mp\_drawing.DrawingSpec(color=(0, 0, 255), thickness=2, circle\_radius=2),

                        mp\_drawing.DrawingSpec(color=(255, 255, 255), thickness=2, circle\_radius=2)

                    )

            save\_preprocessed\_image(black\_image, f"frame\_{frame\_count}.png")

            frame\_count += 1

            processed\_image = preprocess\_image(black\_image, target\_size=(64, 64))

            prediction = model.predict(processed\_image)

            class\_id = np.argmax(prediction)

            predicted\_class = class\_names[class\_id]

            predictions.append(predicted\_class)

        cap.release()

        if predictions:

            most\_common\_class = Counter(predictions).most\_common(1)[0][0]

            if most\_common\_class == 'space':

                result\_string += "  "

            else:

                result\_string += most\_common\_class

print("\nFinal results:\n", result\_string)

correct\_and\_convert\_text\_to\_speech(result\_string)

# output

# conclusion

Application

There exists a large practical application in this hand sign detection, including

1. Enable communication for the deaf and dumb through real-time display of text or speech translation of signed communication.

2. Human-computer interaction: Devices or systems can be controlled by gestures for gaming, virtual reality systems, and smart house technologies, thus making the mode of user interaction seamless and intuitive.

3. Healthcare: Enables telemedicine with remote monitoring of patients and treatment whereby hand movements can be detected and interpreted for the necessity of physical therapy or further consultation with the patient's doctor.

4. Education: Learning aids and software of sign language could be used and engagement of Deaf along with hearing in the interactive learning process could be enabled.

Objective

The development of hand sign detection systems primarily concerns the efficient and correct recognition and interpretation of hand gestures in translating meaningful digital signals. It involves the following:

* Accessibility Improvement – Furthering the possibility or extent of communication and interaction for the users of sign languages.
* User Interface Improvement – More intuitive and responsive interfaces are provided for technology through gesture-based control.
* Research and Development Aide: It will be a tool to aid research and research development to create new applications and technologies based on recognition of hand signs.

Conclusion

The hand sign detection technology seems to be playing a very important role in bridging the communication gaps and thereby enhancing the interaction of the users with technology. The existing systems have been found to offer great improvements in accessing potential and creating more engaging user experiences. However, challenges such as the variability of the hand shape and gesture, environmental conditions, and real-time performance need to be answered. The fulfillment of this criterion is very much required for further development and its widespread adoption.

\*\*Scope for Future\*\*

Some of the propitious improvements which lie ahead in the future for hand gesture detection are:

- \*\*Improvement in accuracy and robustness:\*\* More complex algorithms and improved models will be developed to accurately implement recognition in different conditions and variations.

- \*\*Integration With Emerging Technologies:\*\* Combining emerging hand gesture detection technologies with those of artificial intelligence, augmented reality, and virtual reality in the creation of immersive and adaptive user experiences.

Wearable and Mobile Solutions: The development of wearables and mobile applications has furthered the realization of a practical, non-intrusive solution for detection of useful hand signs in everyday environments. Cross-Language and Cross-Cultural Adaptation: The advances in the detection systems of a wide variety of sign languages inherently capture the cross-cultural adaptations and associated cultural nuances, facilitating global inclusivity and communication.

Future studies and technological advances will no doubt solve the current constraints and open new possibilities for hand sign detection, and therefore these will be further assimilated into digital expression and interaction.

Summary

Existing systems for hand sign detection utilize a range of technologies, from depth sensors and high-precision tracking devices to advanced neural networks and wearable sensors. Each system has its strengths, such as high accuracy, real-time performance, and ease of integration, but also faces limitations related to computational demands, environmental conditions, and user requirements. Continued advancements in machine learning, computer vision, and sensor technologies are expected to address these challenges and improve the effectiveness of hand sign detection systems.

# references