**SIGN LANGUAGE DETECTION**

**A Project Report**

*Submitted in the partial fulfilment for the award of the degree of*

**BACHELOR OF ENGINEERING**

**IN**

**COMPUTER SCIENCE WITH SPECIALIZATION IN**

**ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING**

**Submitted by:**

20BCS6760 HRITVIK MATHUR

20BCS6712 AMISHA KHANNA

**Under the Supervision of:**

**SIDDHARTH KUMAR**



**CHANDIGARH UNIVERSITY, GHARUAN, MOHALI - 140413,**

**PUNJAB**

**March, 2022**



**DECLARATION**

**‘Hritvik Mathur’** and **‘Amisha Khanna’**, students of **‘Bachelor of Engineering in IBM SPECIALISEDARTIFICIAL INTELLIGENCE AND MACHINE LEARNING’**, **session: 2022-23**,

Department of Computer Science and Engineering, Apex Institute of technology, Chandigarh University, Punjab, here by declare that the work presented in this Project Work entitled ‘**Sign Language Detection’** is the outcome of our own bonafide work and is correct to the best of our knowledge and this work has been undertaken taking care of Engineering Ethics and is done under the Supervision of **‘Mr. Siddharth Kumar’**.

**Date:** 01/05/2023

**Hritvik Mathur**

**20BCS6760**

**Amisha Khanna**

**20BCS6712**



# ACKNOWLEDGEMENT

We have taken efforts in this project. However, it would not have been possible without the kind support and help of many individuals and organizations. We would like to extend my sincere thanks to all of them.

We are highly indebted to **Mr. Siddharth Kumar** for hir guidance and constant supervision as well as for providing necessary information regarding the project & also for their support in completing the project.

We would like to express our gratitude towards everyonefor their kind co-operation and encouragement which help me in completion of this project.

We would like to express our special gratitude and thanks to industry persons for giving me such attention and time.

Our thanks and appreciations also go to our colleague in developing the project and people who have willingly helped us out with their abilities.

# Abstract

Research in the field of sign language recognition has made significant advances in recent years. The present achievements provide the basis for future applications with the objective of supporting the integration of deaf people into the hearing society. Translation systems, for example, could facilitate communication between deaf and hearing people in public situations. Further applications, such as user interfaces and automatic indexing of signed videos, become feasible. The current state in sign language recognition is roughly 30 years behind speech recognition, which corresponds to the gradual transition from isolated to continuous recognition for small vocabulary tasks. Research efforts were mainly focused on robust feature extraction or statistical modelling of signs. However, current recognition systems are still designed for signer-dependent operation under laboratory conditions. This paper describes a comprehensive concept for robust visual sign language recognition, which represents the recent developments in this field. The proposed recognition system aims for signer-independent operation and utilizes a single video camera for data acquisition to ensure user-friendliness. Since sign languages make use of manual and facial means of expression, both channels are employed for recognition.

Keywords:

Sign language detection, Sign language processing, Deaf-dumb hearing impaired, intelligent systems

# Table of Contents

|  |  |
| --- | --- |
| Declaration  Acknowledgement  Abstract | i  ii  iii |
| 1. List of Figures | 1 |
| 2. List of Abbreviations | 3 |
| 3.    Introduction |  |
| 1.1  Problem Definition | 4 |
| 1.2 Project Overview | 5 |
| 1.3 Hardware Specification | 6 |
| 1.4 Software Specification | 6 |
| 4.    Literature Survey | 7 |
| 2.1 Existing System | 9 |
| 2.2 Proposed System | 10 |
| 2.3 Literature Review Summary | 11 |
| 5.    Problem Formulation | 12 |
| 6.    Research Objective | 14 |
| 7.    Methodologies | 15 |
| 8. Result / Analysis | 45 |
| 9.    Conclusion | 58 |
| 10.    Reference | 59 |

# LIST OF FIGURES

|  |  |
| --- | --- |
| Figure 1 | Deaf Human |
| Figure 2 | Lack of communication |
| Figure 3 | Indian Sign Language |
| Figure 4 | Flowchart |
| Figure 5 | Flow diagram of proposed system |
| Figure 6 | American Sign Language |
| Figure 7 | Predicting ISL using dataset |
| Figure 8 | Binary image |
| Figure 9 | Mask |
| Figure 10 | Dataset |
| Figure 11 | Extracting features |
| Figure 12 | Input image |
| Figure 13 | Extracted features |
| Figure 14 | Sample Data |
| Figure 15 | Architecture of CNN |
| Figure 16 | Flowchart |
| Figure 17 | Snapshot of proposed System |
| Figure 18 | System GUI |

# LIST OF FIGURES

|  |  |
| --- | --- |
| Figure 19 | User panel using Tkinker |
| Figure 20 | Class wise accuracy table |
| Figure 21 | Accuracy graph of CNN |
| Figure 22 | Accuracy Table |
| Figure 23 | Performance Metrices Table |
| Figure 24 | Comparison of different techniques |

# LIST OF ABRIVATIONS

|  |  |
| --- | --- |
| ISL | Indian Sign Language |
| ASL | American Sign Language |
| CNN | Convocational Neural Network |
| BOW | Box Of Words |
| SVM | Support Vector Machine |
| RNN | Recurrent Neural Network |
| KNN | K- Nearest Neighbour |
| GUI | Graphical User Interface |
| ML | Machine Learning |

# 1. INTRODUCTION

## 1.1 Problem Definition

Deaf and hard-of-hearing persons, as well as others who are unable to communicate verbally, utilise sign language to communicate within their communities and with others. Sign languages are a set of present languages that communicate information using a visual-manual modality. The dilemma of real-time finger-spelling recognition in Sign Language is discussed. We gathered a dataset for identifying 36 distinct gestures (alphabets and numerals) and a dataset for typical hand gestures in ISL created from scratch using webcam images. The system accepts a hand gesture as input and displays the identified character on the monitor screen in real time. This project falls under the category of human-computer interaction (HCI) and tries to recognise multiple alphabets (a-z), digits (0-9) and several typical ISL hand gestures. To apply Transfer learning to the problem, we used a Pre-Trained SSD Mobile net V2 architecture trained on our own dataset. In the vast majority of situations, we constructed a robust model that consistently classifies Sign language.



Fig 1: Deaf Human

This project falls within the HCI (Human Computer Interface) sector and seeks to recognise multiple alphabets (a-z), digits (0-9) and several typical ISL family hand motions such as Thank you, Hello, and so on. Hand-gesture recognition is a difficult problem, and ISL recognition is particularly difficult owing to the use of both hands. Many studies have been done in the past employing sensors (such as glove sensors) and various image processing techniques (such as edge detection, Hough Transform, and so on), but they are quite costly, and many people cannot afford them.

## 1.2 Problem Overview

Many people in India are deaf or hard of hearing, thus they communicate with others using hand gestures. However, aside from a small group of people, not everyone is familiar with sign language, and they may need an interpreter, which may be complex and costly. The goal of this research is to build software that can anticipate ISL alphanumeric hand movements in real time, bridging the communication gap.

Sign language is largely used by the disabled, and there are few others who understand it, such as relatives, activists, and teachers at SekolahLuarBiasa (SLB). Natural gestures and formal cues are the two types of sign language[1]. The natural cue is a manual (hand-handed) expression agreed upon by the user (conventionally), recognised to be limited in a particular group (esoteric), and a substitute for words used by a deaf person (as opposed to body language). A formal gesture is a cue that is established deliberately and has the same language structure as the community's spoken language.

Sign language is a visual language. It mainly consists of 3 major components:

1.Fingerspelling: Spell out words character by character, and word level association which involves hand gestures that convey the word meaning. The static Image Dataset is used for this purpose.

2.World-level sign vocabulary: The entire gesture of words or alphabets is recognized through video classification. (Dynamic Input / Video Classification)

3.Non-manual features: Facial expressions, tongue, mouth, body positions

## 1.3 Hardware Specification

1.Interface: Jupyter notebook for inserting python libraries in a notebook format, it is typically a python code where we can easily estimate our data sets model in one single notebook.

2.Operating System Environment: Windows 10

3.Hardware Environment: RAM- 16GB ,GRAPHIC CARD – 6GB , ROM-1060TB

## 1.4 Software Specification

Software: Python (3.7.4), Anaconda(2019-0.7) ,IDE (Jupyter), Numpy (version 1.16.5), cv2 (openCV) (version 3.4.2) , Tensorflow (version 2.0.0) , Github , Virtual Studio (2022) ,CUDA(10.1) and CuDNN(7.6) (For NIVIA GPU for faster training model) ,Protoc

# 2. LITERATURE SURVEY

Starner and Pentland[10] provided one of the early studies on sign language recognition. They demonstrated a real-time hidden Markov model-based system that detected sentence level American Sign Language (ASL) movements with the help of a webcam. They described two experiments: the first uses a desk-mounted camera to view the user, while the second uses a camera embedded in the user's cap.

The authors in paper [3] presented a system for the automatic translation of the gestures of the manual alphabets in Arabic Sign Language. This system made use of images of the gesture as input which were then processed and converted into a set of features that comprised of some length measures which indicated the fingertip’s position. The subtractive clustering algorithm and the least-squares estimator were used for classification. The system achieved an accuracy of 95.55%.

In [4] Nadia R. Albelwi and Yasser M. Alginahi proposed a real-time Arabic Sign Language system where a video camera was used to capture real-time video as an input to the system. The authors used a Haar-like algorithm to track the hand in the video frames and applied preprocessing techniques like skin detection and size normalization to extract the region of interest. To obtain the feature vectors, Fourier Transformation is applied to the resultant images which are transformed into the frequency domain. The classification is performed using the k-Nearest Neighbor (KNN) algorithm and the system achieves an accuracy of 90.55%.

In [5] Balakrishnan, G., P. S. Rajam, et al., proposed a system that converts a set of 32 combinations of the binary number, which represents the UP and DOWN positions of the five fingers into decimal. The binary numbers are first converted into a decimal form by using the binary-decimal conversion algorithm and then the decimal numbers are converted to their corresponding Tamil letters. Static images of the gesture were used as the input to the system where a canny-edge detection algorithm was applied to extract the edges of the palm and Euclidean Distance was applied to identify the position of the fingers. The system achieved an accuracy of 98.75%.

The authors in paper [6] proposed a system employing bspline approximation to develop a novel vision-based recognition system for Indian sign language alphabets and digits. By using the Maximum Curvature Points (MCPs) as Control points, their technique approximates the extracted boundary from the region of interest to a B-Spline curve. The B-spline curve is then smoothed iteratively, resulting in the extraction of Key Maximum Curvature Points (KMCPs), which are the major contributors to the gesture shape. As a result, the spatial coordinates of the KMCPs in the 8 Octant Regions of the 2D Space that are given for classification yield a translation and scale-invariant feature vector. The accuracy of numbers was 93.2 percent, and the accuracy of alphabets was 91.83 percent.

In paper [7], the authors proposed a system that can recognize and convert ISL gestures from a video feed into English voice and text. They did this by segmenting the shapes in the video stream using several image processing techniques such as edge detection, wavelet transform, and picture fusion. Ellipsoidal Fourier descriptors were used to extract shape features, while PCA was utilized to optimize and reduce the feature set. The fuzzy inference system was used to train the system, which resulted in a 91% accuracy.

In paper [8], the authors suggested a method for automatically recognizing Indian sign language gestures. The proposed method employs digital image processing techniques and uses the YCbCr color space for hand detection, with the input image being transformed beforehand. Distance transformation, Projection of distance transformation coefficients, Fourier descriptors, and feature vectors are some of the techniques used to extract the features. An artificial neural network was used to classify the data, and the recognition rate was 91.11 percent.

## 2.1 Existing System

In existing system, the module was developed for dumb person using flex sensor, there user hand is attached with the flex sensors. On this module the flex sensor reacts on bend of each finger individually. By taking that value controller starts to react with speech, each flex sensor holds unique voice stored in APR Kit and for each sign it will play unique voice. And in other existing system, the work is done only for some alphabets and not for the words or sentences, and accuracy obtained is very low.

**Limitations of existing system**

* In existing system it’s restricted to only 10 voice announcements it may reduce product capacity
* One of the major problems of the existing system is Dumb person should always carry the hardware with him
* User can’t do any other work with flex sensor on fingers and also sensors should be placed straight
* The controller may think that the user is giving command and finally it may result in unwanted results and less hardware lifetime

## 2.2 Proposed System

In the proposed system the unable or dumb person should provide a gesture or sign image to the system. The system evaluates the sign input with matlab image processing technique and classifies the input to the recognized identification. Later it initiates the voice media through the system when the input image matches with the given dataset. And the output will be shown in the text format too. This is a prototype to develop the concept of converting the sign language to speech and text. The aim of this paper is to provide an application to the society to establish the ease of communication between the deaf and mute people by making use of image processing algorithm.

**Advantages of proposed system**

* When comparing with existing system user can give more signs
* The module provides two-way communications which helps in easy interaction between the normal people and disables
* Easy to Interface
* Flexible

## 2.3 Literature Review Summary

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Year and**  **Citation** | **Article/ Author** | **Tools/ Software** | **Technique** | **Source** | **Evaluation Parameter** |
| 2022 | Real Time Sign Language Recognition System for Hearing and Speech Impaired People | Python Machine Learning | CNN | http://surl.li/fhxbe | Accuracy of 80% |
| 2022 | Deepsign: Sign Language Detection and Recognition | Python Deep Learning | feedback-based learning models | https://rb.gy/ifvz7j | 87% Accuracy |
| 2019 | Sign Language Detection “in the Wild” | Python Machine Learning | RNN | https://rb.gy/tqutsj | Precision of 83% |
| 2021 | ML Based Sign Language Recognition System | Python Machine Learning | KNN | https://rb.gy/qejolc | 65% Accuracy. |
| 2017 | Machine Learning Techniques for Indian Sign Language Recognition | Python Machine Learning | ML Algorithms | https://rb.gy/gakute | 90% |

# 3. PROBLEM FORMULATION

Conversing with people having a hearing disability is a major challenge. Deaf and Mute people use hand gesture sign language to communicate, hence normal people face problems in recognizing their language by signs made. Hence there is a need for systems that recognize the different signs and conveys the information to normal people.

The solution is to develop a translator that can detect sign language used by a disabled person, and then feed that sign into a machine-learning algorithm called transfer learning, which is then detected by the neural network and translated on the screen so that a normal person can understand what the sign is saying.



Fig 2: Lack of Communication

It's a lot easier now, thanks to speech to text and translators. But what about individuals who are unable to speak or hear?   The main goal of this project is to create an application that can assist persons being unable to speak or hear. The language barrier is also a very significant issue. Hand signals and gestures are used by people who are unable to speak. Ordinary people have trouble comprehending their own language. As a result, a system that identifies various signals and gestures and relays information to ordinary people is required. It connects persons who are physically handicapped with others who are not.

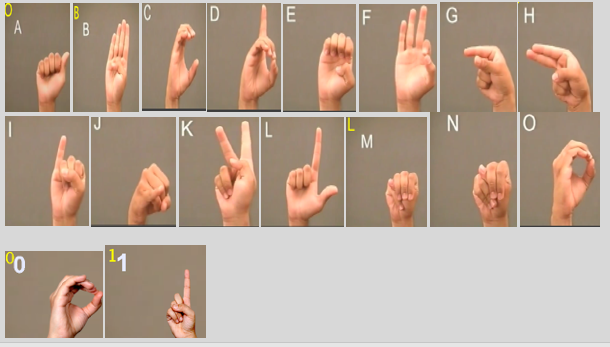


Fig 3: Indian Sign Language

Many firms are creating solutions for deaf and hard of hearing persons, but not everyone can afford them. Some are very pricey for ordinary middle-class individuals to bring.

# 4. OBJECTIVES

The proposed work is aimed to carry out work leading to the development of an approach for SIGN LANGUAGE DETECTION MODEL. More than 360 million of world population suffers from hearing and speech impairments [3]. Sign language detection is a project implementation for designing a model in which web camera is used for capturing images of hand gestures which is done by open cv.

After capturing images, labelling of images are required and then pre trained model SSD Mobile net v2 is used for sign recognition. Thus, an effective path of communication can be developed between deaf and normal audience. Three steps must be completed in real time to solve our problem:

1. Obtaining footage of the user signing is step one (input).

2. Classifying each frame in the video to a sign.

3. Reconstructing and displaying the most likely Sign from classification scores (output).

# 5. METHODOLOGY

The following methodology will be followed to achieve the objectives defined for proposed research work:

Phase1: Searching research papers and collecting data of Sign Language Detection, software requirements.

Phase2: Implementation of code of Sign Language Detection and Data Gathering and Train the Recognizer

Phase3: Implementation of code of Model

Phase4: Finalize Project and All Documentation of project.

* Fundamental steps in image processing are:

1. Image acquisition: to acquire a digital image

2. Image pre-processing: to improve the image in ways that increases the chances for success of the other processes.

3. Image segmentation: to partitions an input image into its constituent parts of objects.

4. Image description: to extract the features that result in some quantitative information of interest of features that are basic for differentiating one class of objects from another.

5. Image recognition: to assign a label to an object based on the information provided by its description.

6. Image segmentation: to convert the input data to a from suitable for computer processing.

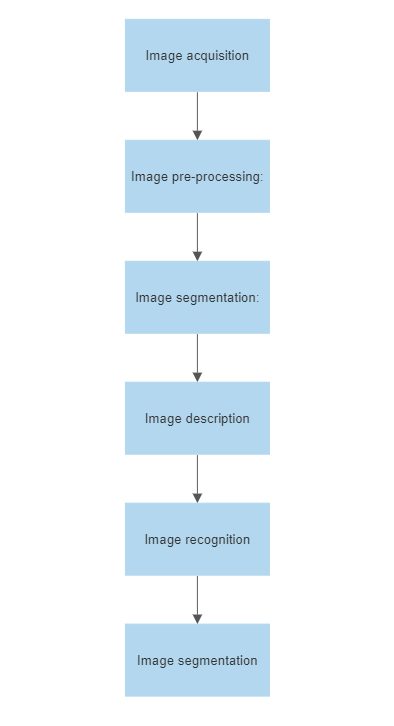


Fig 4: Flowchart

The research paper being discussed presents a sign language detection system that utilizes computer vision techniques, specifically employing a convolutional neural network (CNN) for gesture recognition. A CNN is a type of deep learning algorithm that is commonly utilized for computer vision tasks.

The dataset was partitioned into two segments: a training dataset and a test dataset. The training dataset was utilized to train the CNN, while the test dataset was used to evaluate the system's performance. The CNN was trained on the training dataset using backpropagation to update the network's weights. The system's accuracy was calculated by evaluating it on the test dataset.

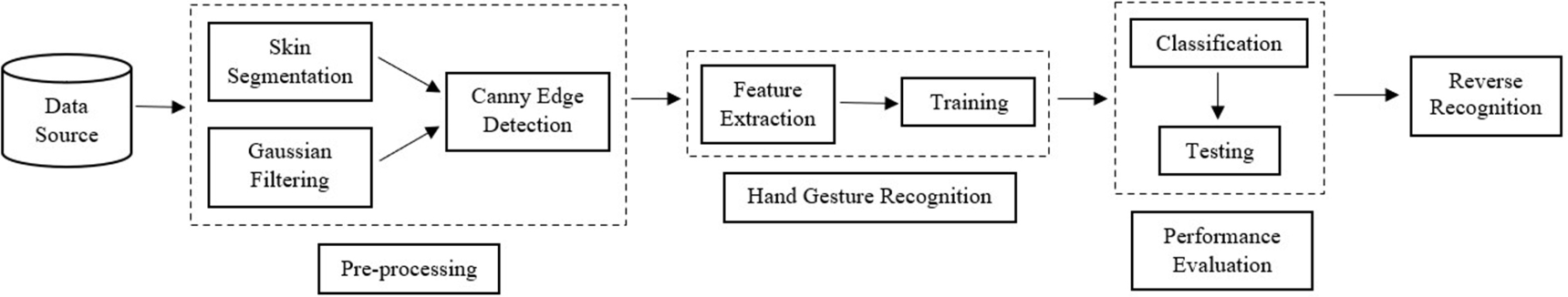


Fig 5 -: Flow Diagram of Proposed Method

**Code to Create Dataset:**

def cd\_main():

    # importing necessary files

    import cv2

    import imutils

    import numpy as np

    import os

    from os import path

    import tkinter as tk

    from tkinter import  messagebox

    bg = None

    #To find the running average over the background

    def run\_avg(image,aweight):

        nonlocal bg

        #initialize the background

        if bg is None:

            bg=image.copy().astype("float")

            return

        cv2.accumulateWeighted(image,bg,aweight)

    # Segment the egion of hand

    def extract\_hand(image,threshold=25):

        nonlocal bg

        diff=cv2.absdiff(bg.astype("uint8"),image)

        thresh=cv2.threshold(diff,threshold,255,cv2.THRESH\_BINARY)[1]

        (\_,cnts,\_)=cv2.findContours(thresh,cv2.RETR\_EXTERNAL,cv2.CHAIN\_APPROX\_SIMPLE)

        if(len(cnts)==0):

            return

        else:

            max\_cont=max(cnts,key=cv2.contourArea)

            return (thresh,max\_cont)

    def n(x):

        pass

    aWeight=0.5

    cam=cv2.VideoCapture(0)

    #t,r,b,l=100,350,228,478

    t,r,b,l=100,350,325,575

    num\_frames=0

    cur\_mode=None

    count=0

    limit=500

    method=1

    #print("Enter the method: (1 for keeping bg still and 2 for skin extraction")

    #method=int(input())

    option=messagebox.askquestion('Select option','Choose default method ?')

    if option=='yes':

        method=2

    else:

        method=1

    if method==2:

        cv2.namedWindow('Tracking', cv2.WINDOW\_NORMAL)

        cv2.resizeWindow("Tracking", 640, 480)

        cv2.createTrackbar("LH", "Tracking", 0, 255, n)

        cv2.createTrackbar("LS", "Tracking", 0, 255, n)

        cv2.createTrackbar("LV", "Tracking", 0, 255, n)

        cv2.createTrackbar("UH", "Tracking", 255, 255, n)

        cv2.createTrackbar("US", "Tracking", 32, 255, n)

        cv2.createTrackbar("UV", "Tracking", 255, 255, n)

    while(cam.isOpened()):

        \_,frame=cam.read()

        if frame is not None:

            frame=imutils.resize(frame,width=700)

            frame=cv2.flip(frame,1)

            clone=frame.copy()

            # height,width=frame.shape[:2]

            roi=frame[t:b,r:l]

            if method ==1:

                gray = cv2.cvtColor(roi, cv2.COLOR\_BGR2GRAY)

                gray = cv2.GaussianBlur(gray, (7, 7), 0)

                if(num\_frames<30):

                    run\_avg(gray,aWeight)

                    cv2.putText(clone, "Keep the Camera still.", (10, 100), cv2.FONT\_HERSHEY\_COMPLEX, 0.8, (0, 0, 0))

                else:

                    cv2.putText(clone, "Press esc to exit.", (10, 200), cv2.FONT\_HERSHEY\_COMPLEX, 0.5, (0, 0, 0))

                    cv2.putText(clone, "Keep the Camera still.", (10, 50), cv2.FONT\_HERSHEY\_COMPLEX, 0.8, (0, 0, 0))

                    cv2.putText(clone, "Put your hand in the rectangle", (10, 100), cv2.FONT\_HERSHEY\_COMPLEX, 0.5, (0, 0, 0))

                    cv2.putText(clone, "Press the key of the sample", (10, 150), cv2.FONT\_HERSHEY\_COMPLEX, 0.5, (0, 0, 0))

                    hand=extract\_hand(gray)

                    if hand is not None:

                        thresh, max\_cont = hand

                        mask = cv2.drawContours(clone, [max\_cont + (r, t)], -1, (0, 0, 255))

                        cv2.imshow("Threshold", thresh)

                        mask = np.zeros(thresh.shape, dtype="uint8")

                        cv2.drawContours(mask, [max\_cont], -1, 255, -1)

                        mask = cv2.medianBlur(mask, 5)

                        mask = cv2.addWeighted(mask, 0.5, mask, 0.5, 0.0)

                        kernel = np.ones((5, 5), np.uint8)

                        mask = cv2.morphologyEx(mask, cv2.MORPH\_CLOSE, kernel)

                        res = cv2.bitwise\_and(roi, roi, mask=mask)

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

                        cv2.imshow("Extracted", res)

                        high\_thresh, thresh\_im = cv2.threshold(res, 0, 255, cv2.THRESH\_BINARY + cv2.THRESH\_OTSU)

                        lowThresh = 0.5 \* high\_thresh

                        res = cv2.Canny(res, lowThresh, high\_thresh)

            if method==2:

                hsv = cv2.cvtColor(roi, cv2.COLOR\_BGR2HSV)

                lh = cv2.getTrackbarPos("LH", "Tracking")

                ls = cv2.getTrackbarPos("LS", "Tracking")

                lv = cv2.getTrackbarPos("LV", "Tracking")

                uh = cv2.getTrackbarPos("UH", "Tracking")

                us = cv2.getTrackbarPos("US", "Tracking")

                uv = cv2.getTrackbarPos("UV", "Tracking")

                l\_b = np.array([lh, ls, lv])

                u\_b = np.array([uh, us, uv])

                cv2.putText(clone, "Put your hand in the rectangle", (10, 50), cv2.FONT\_HERSHEY\_COMPLEX, 0.5,(0, 0, 0))

                cv2.putText(clone, "Adjust the values using trackbar", (10, 100), cv2.FONT\_HERSHEY\_COMPLEX, 0.5, (0, 0, 0))

                cv2.putText(clone, "Press the key of the sample", (10, 150), cv2.FONT\_HERSHEY\_COMPLEX, 0.5, (0, 0, 0))

                cv2.putText(clone, "Press esc to exit.", (10, 200), cv2.FONT\_HERSHEY\_COMPLEX, 0.5, (0, 0, 0))

                mask = cv2.inRange(hsv, l\_b, u\_b)

                cv2.imshow('mask', mask)

                mask = cv2.bitwise\_not(mask)

                mask = cv2.medianBlur(mask, 5)

                mask = cv2.addWeighted(mask, 0.5, mask, 0.5, 0.0)

                kernel = np.ones((5, 5), np.uint8)

                mask = cv2.morphologyEx(mask, cv2.MORPH\_CLOSE, kernel)

                res = cv2.bitwise\_and(roi, roi, mask=mask)

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

                cv2.imshow('res', res)

                high\_thresh, thresh\_im = cv2.threshold(res, 0, 255, cv2.THRESH\_BINARY + cv2.THRESH\_OTSU)

                lowThresh = 0.5 \* high\_thresh

                res = cv2.Canny(res, lowThresh, high\_thresh)

            # hand = cv2.bitwise\_and(gray, gray, mask=thresh)

            # cv2.imshow("Hand", hand)

            # res = cv2.Canny(hand, lowThresh, high\_thresh)

            # cv2.imshow("Hand res", res)

            # v = np.median(res)

            # sigma=0.33

            if cur\_mode!=-1 and cur\_mode!=255 and cur\_mode is not None:

                file\_path = 'Saved Dataset\\'+str(chr(cur\_mode))

                if not path.exists(file\_path):

                    os.makedirs(file\_path)

                if(count<=limit):

                    cv2.imwrite(file\_path+'\\'+str(count)+'.jpg',res)

                    print(count)

                    if(count==limit):

                        print("Completed")

                count+=1

            cv2.rectangle(clone,(l,t),(r,b),(0,255,0),2)

            num\_frames+=1

            cv2.imshow("Video Feed",clone)

        else:

            messagebox.showerror("error","Can't grab frame")

            break

        k=cv2.waitKey(1)& 0xFF

        if (k==27):

           break

        if(k!=-1 and k!=255 and k!=cur\_mode):

            cur\_mode=k

            count = 0

    cam.release()

    cv2.destroyAllWindows()

#cd\_main()

1. **Collecting a dataset:** You can collect a dataset of hand sign images or videos by either capturing them using a camera or using an existing dataset. Some popular datasets for hand sign recognition include American Sign Language (ASL) alphabet dataset, Indian Sign Language (ISL) gesture dataset, and Hand Gesture Recognition Database (HGRDB) dataset.



Fig 6: American Sign Language

Collecting data is a crucial aspect of research in all fields as it forms the foundation for training any model. There is lack of standard datasets for Indian sign language. To address this issue, we manually constructed a dataset as part of this project.

We began by capturing videos using a webcam. We considered alphabets and 10 numeric signs from three individuals. To add variation to the dataset, two options were employed to capture the images. The first method involves default skin segmentation on the image and can be used with a plain colour background.

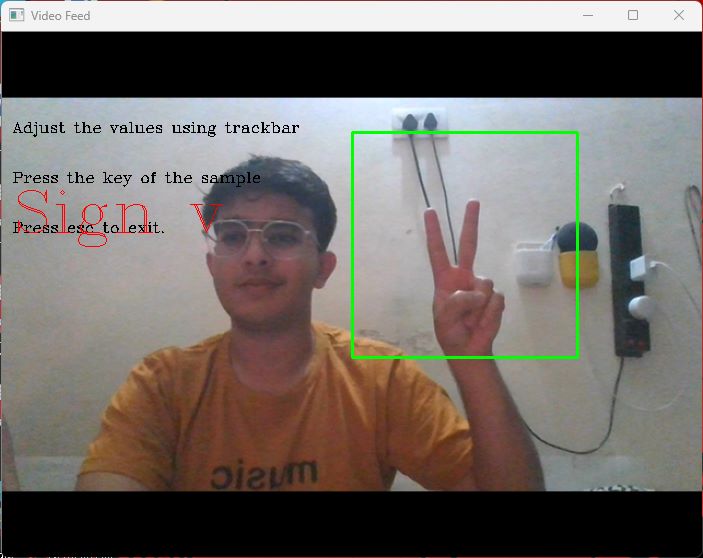


Fig 7: Predicting ISL using Dataset

The second method we utilized involved running averages, where any new object after the initial frames was considered background, making the extraction process easy. Both of these approaches were taken into account.

Signs were converted into different segments. The frames produced had a resolution of 250\*250 to reduce pre-processing computational power requirements.

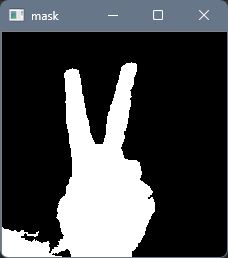
 

Fig 8: Binary Image Fig 9: Mask

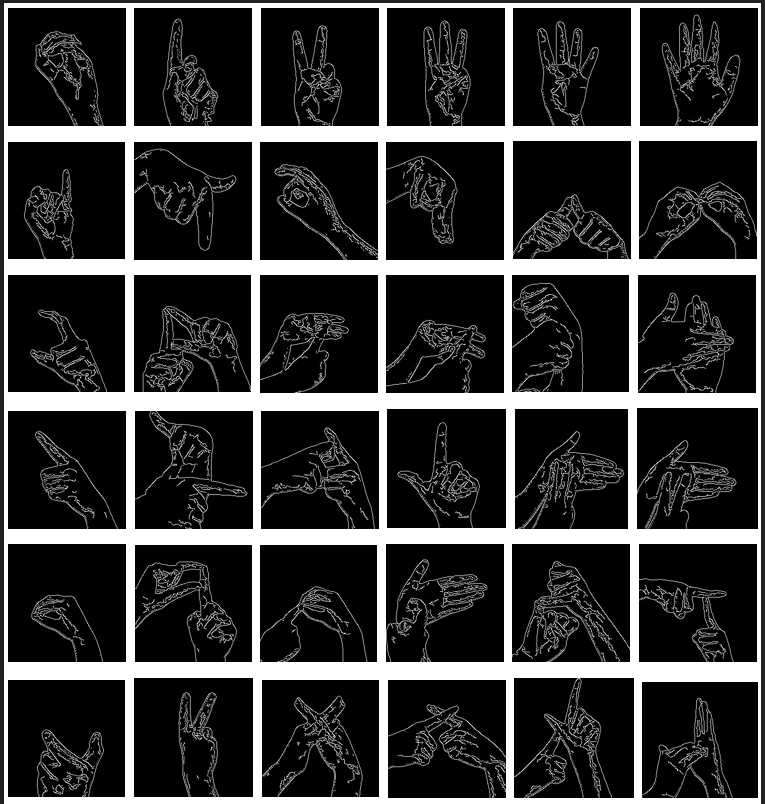


Fig 10: Dataset

1. **Pre-processing:** You can use image processing techniques to pre-process the dataset. This may involve resizing the images, normalizing the pixel values, and converting them to grayscale or colour.

Input is made ready to go for feature detection and extraction.

Preprocessing in sign language detection using CNN is an essential step that involves preparing the input data for the neural network to improve its accuracy and efficiency. Some common preprocessing techniques that can be used in sign language detection using CNN include:

Data Cleaning: Data cleaning involves removing any unnecessary or irrelevant information from the input data. This can include removing noise or outliers, as well as normalizing the data.

Data Augmentation: Data augmentation is a technique that involves creating new training data by applying transformations such as rotation, scaling, or cropping to the existing data. This helps to increase the size of the training data set and improve the model's performance.

Data Resizing: In sign language detection using CNN, resizing the input images to a specific size is essential to ensure that the images have the same dimensions for the neural network to process them.

Data Normalization: Data normalization is a technique used to scale the input data to a common range. This can improve the model's performance and reduce the training time.

1. **Extraction of features:** We extract features from the pre-processed images

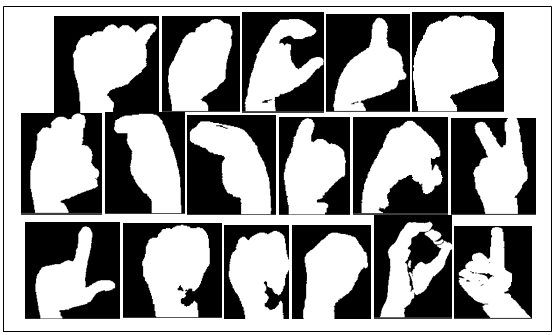


Fig 11: Extracting Features

In this phase, the researchers developed a Bag of Visual Words model for the image classification task.

The BOVW model is a popular image classification technique that is adapted from the Bag of Words (BOW) model used in natural language processing (NLP). In the BOW model, the frequency of words in a text document is used to generate a histogram of keywords. The BOVW model uses a similar approach, but instead of words, image features are used as the vocabulary.

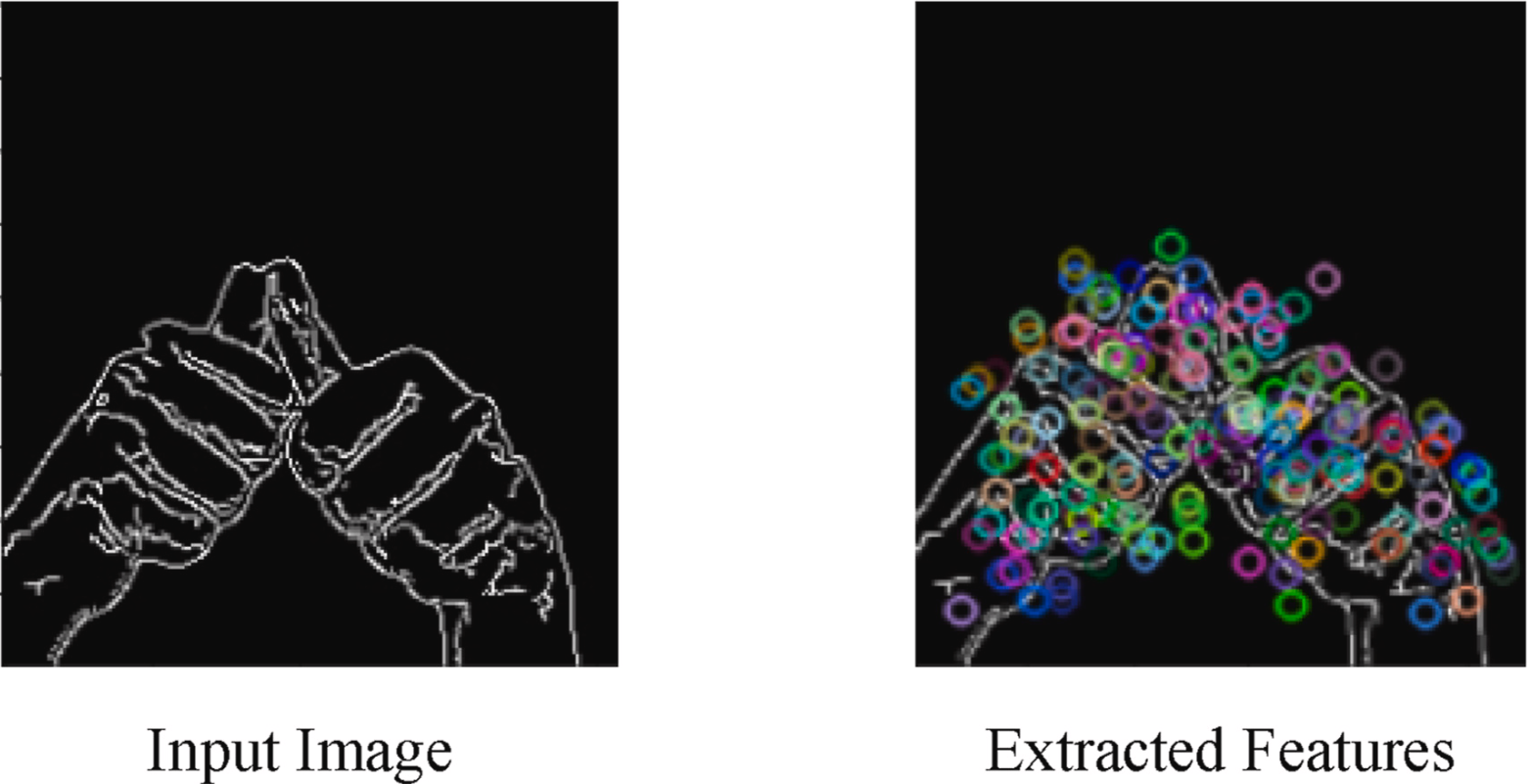


Fig 12: Input Image Fig 13: Extracted features

1. **Training the model:** We train a ML algorithm such as SVM, KNN etc on the extracted features. You can use libraries such as scikit-learn or TensorFlow for building the machine learning model.

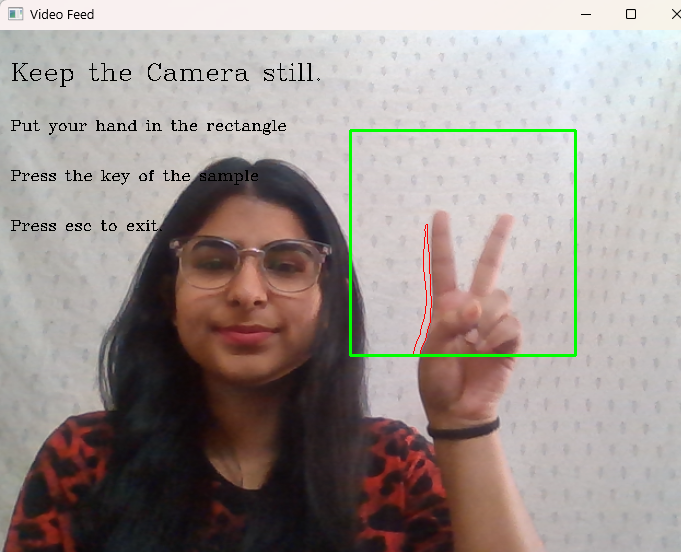


Fig 14: Sample Data

**Requirements:**

contrib==0.3.0

google-auth==1.9.0

google-auth-oauthlib==0.4.1

google-pasta==0.1.8

graphviz==0.14

gTTS==2.1.1

gTTS-token==1.1.3

imageio==2.8.0

importlib-metadata==1.3.0

imutils==0.5.3

ipykernel==5.1.3

ipython==7.10.2

joblib==0.14.1

jupyter==1.0.0

jupyter-client==5.3.4

jupyter-console==6.0.0

jupyter-core==4.6.1

Keras==2.3.1

Keras-Applications==1.0.8

Keras-Preprocessing==1.1.0

matplotlib==3.1.2

notebook==6.1.5

numpy==1.16.4

opencv-contrib-python==3.4.2.16

opencv-python==3.4.2.16

pandas==0.25.3

pickleshare==0.7.5

Pillow>=7.1.0

playsound==1.2.2

pyttsx3==2.88

pywin32==227

scikit-learn==0.22.2.post1

scikit-video==1.1.11

scipy==1.4.1

seaborn==0.10.1

sklearn==0.0

SpeechRecognition==3.8.1

tensorboard==1.15.0

tensorflow==1.15.2

tensorflow-estimator==1.15.1

tflearn==0.3.2

ttkthemes==3.1.0

virtualenv==20.0.4

virtualenvwrapper-win==1.2.6

XlsxWriter==1.2.9

zipp==0.6.0

**Classification**:

***Naïve Bayes:***

Naive Bayes is a ML algorithm that can be used in recognizing sign language. Sign language recognition involves interpreting and recognizing.

In this, the probability of each feature (or input) is calculated independently of the other features. In the context of sign language recognition, the features could be hand position, hand shape, and movement trajectory.

To train a naive Bayes classifier for sign language recognition, a dataset of sign language gestures with known labels is needed. During the training phase, the classifier calculates the probability of each feature given each label. These probabilities are stored in the classifier and used to make predictions on new input.

When a new sign language gesture is presented to the classifier, the probabilities of each feature are calculated and used to determine the probability of each label.

Overall, naive Bayes is a useful algorithm for sign language recognition because it can handle multiple features and can make predictions quickly. However, it is important to note that the "naive" assumption of independence between features may not always hold in real-world scenarios.

***Support Vector Machine: (SVM)***

They are also ML algorithm that can also be used in sign language recognition systems.

In the context of sign language recognition, SVMs can be used to classify hand gestures based on various features such as hand shape, hand orientation, and movement direction

In order to utilize SVM for sign language recognition, a dataset that contains labelled sign language gestures is necessary. The features of these gestures are then extracted and used to train the SVM. During the training process, the SVM aims to locate the most optimal hyperplane that divides the various classes of gestures based on their feature values. Once the SVM is fully trained, it can be employed to predict the label of a new input gesture.

One advantage of using SVMs in sign language recognition is that they can handle high-dimensional feature spaces and can find a solution even when the data points are not linearly separable. However, SVMs can be computationally expensive, especially when the number of features or data points is large.

Overall, SVMs can be a powerful tool for sign language recognition, especially when dealing with complex data. However, the performance of the algorithm depends heavily on the choice of kernel function and tuning of the hyperparameters.

***Convolutional Neural Networks (CNN)***

CNN are DL algorithm that has gained significant popularity in recent years, particularly for image recognition tasks such as sign language recognition. CNNs are well-suited for handling images as inputs and can automatically extract relevant features from them, making them particularly effective for this type of task.

In the context of sign language recognition, CNNs can be used to classify hand gestures based on various features such as hand shape, orientation, and movement. This process is repeated multiple times to extract higher-level features, which are then fed into a fully connected layer for classification.

To train a CNN for sign language recognition, a large dataset of labelled sign language gestures is needed. The gestures are typically represented as images, and the features of the gestures are automatically learned by the CNN during the training phase. The network is trained using backpropagation, where the weights of the filters are adjusted to minimize the classification error.

One advantage of using CNNs in sign language recognition is that they can handle complex, high-dimensional feature spaces and can learn to extract features that are relevant for the classification task. Overall, CNNs can be a powerful tool for sign language recognition, especially when dealing with complex and varied sign language gestures.

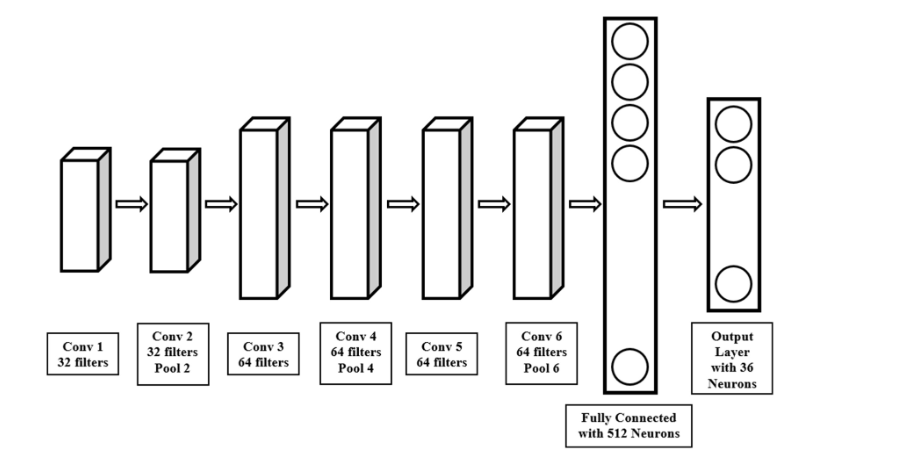


Fig 15: Architecture of CNN

**CODE TO DO PREDICTION USING CNN:**

# Importing the necessary libraries

import os

import cv2

import random

import tensorflow as tf

from tensorflow import keras

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.preprocessing import LabelEncoder

import pickle

import keras

from keras.models import Sequential

from keras.callbacks import \*

from keras.layers import \*

from keras.preprocessing.image import ImageDataGenerator

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

from keras.callbacks import TensorBoard

import matplotlib.pyplot as plt

from keras.utils import to\_categorical

# Loading images

# Converting images to an size of (100,100)

def load\_images(folder):

  train\_data=[]

  for label in os.listdir(folder):

    print(label, " Started!")

    path=folder+'/'+label

    for img in os.listdir(path):

      img=cv2.imread(path+'/'+img,cv2.IMREAD\_GRAYSCALE)

      new\_img=cv2.resize(img,(100,100))

      if new\_img is not None:

        train\_data.append([new\_img,label])

    print(label, " ended!")

  return train\_data

# Loading the train images and their corresponding labels

path='/ISL Datasets/Train-Test/Train'

train\_data=load\_images(path)

from google.colab import drive

drive.mount('/content/drive')

# Loading the test images and their corresponding labels

path='ISL Datasets/Train-Test/Test'

test\_data=load\_images(path)

# Shuffling the data

random.shuffle(train\_data)

random.shuffle(test\_data)

# Seperating features and labels

train\_images=[]

train\_labels=[]

test\_images=[]

test\_labels=[]

for feature, label in train\_data:

  train\_images.append(feature)

  train\_labels.append(label)

for feature, label in test\_data:

  test\_images.append(feature)

  test\_labels.append(label)

#print(len(train\_images))

#print(len(test\_images))

# Converting images list to numpy array

train\_images=np.array(train\_images)

test\_images=np.array(test\_images)

train\_images=train\_images.reshape((-1,100,100,1))

test\_images=test\_images.reshape((-1,100,100,1))

train\_images.shape

#test\_images.shape

# Changing the datatype and Normalizing the data

train\_images= train\_images.astype('float32')

test\_images = test\_images.astype('float32')

train\_images=train\_images/255.0

test\_images=test\_images/255.0

# Encoding the label values

le=LabelEncoder()

le.fit\_transform(train\_labels)

le.fit\_transform(test\_labels)

train\_labels\_label\_encoded=le.transform(train\_labels)

test\_labels\_label\_encoded=le.transform(test\_labels)

test\_labels\_label\_encoded

# One hot encoding

train\_labels\_one\_hot = to\_categorical(train\_labels\_label\_encoded)

test\_labels\_one\_hot = to\_categorical(test\_labels\_label\_encoded)

print('Original label 0 : ', train\_labels[0])

print('After conversion to categorical ( one-hot ) : ', train\_labels\_one\_hot[0])

print(test\_labels\_one\_hot.shape)

# Developing the Convolutional Neural Network Model

input\_shape=(100,100,1)

n\_classes=36

def create\_model():

    model = Sequential()

    # The first two layers with 32 filters of window size 3x3

    model.add(Conv2D(32, (3, 3), padding='same', activation='relu', input\_shape=input\_shape))

    model.add(Conv2D(32, (3, 3), activation='relu'))

    model.add(MaxPooling2D(pool\_size=(2, 2)))

    model.add(Dropout(0.25))

    model.add(Conv2D(64, (3, 3), padding='same', activation='relu'))

    model.add(Conv2D(64, (3, 3), activation='relu'))

    model.add(MaxPooling2D(pool\_size=(2, 2)))

    model.add(Dropout(0.25))

    model.add(Conv2D(64, (3, 3), padding='same', activation='relu'))

    model.add(Conv2D(64, (3, 3), activation='relu'))

    model.add(MaxPooling2D(pool\_size=(2, 2)))

    model.add(Dropout(0.25))

    model.add(Flatten())

    model.add(Dense(512, activation='relu'))

    model.add(Dropout(0.5))

    model.add(Dense(n\_classes, activation='softmax'))

    return model

model=create\_model()

batch\_size=256

epochs=100

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

model.summary()

history = model.fit(train\_images, train\_labels\_one\_hot, batch\_size=batch\_size, epochs=epochs, verbose=1, validation\_data=(test\_images, test\_labels\_one\_hot))

model.evaluate(test\_images, test\_labels\_one\_hot)

# Visualizing loss

plt.figure(figsize=[8,6])

plt.plot(history.history['loss'],'r',linewidth=2.0)

plt.plot(history.history['val\_loss'],'b',linewidth=2.0)

plt.legend(['Training loss', 'Validation Loss'],fontsize=15)

plt.xlabel('Epochs ',fontsize=16)

plt.ylabel('Loss',fontsize=16)

plt.title('Loss Curves',fontsize=16)

# Visualizing accuracy

plt.figure(figsize=[8,6])

plt.plot(history.history['accuracy'],'r',linewidth=2.0)

plt.plot(history.history['val\_accuracy'],'b',linewidth=2.0)

plt.legend(['Training Accuracy', 'Validation Accuracy'],fontsize=15)

plt.xlabel('Epochs ',fontsize=16)

plt.ylabel('Accuracy',fontsize=16)

plt.title('Accuracy Curves',fontsize=16)

import pickle

file\_name='/Saved Files/CNN'

outfile=open(file\_name,'wb')

pickle.dump(model,outfile)

outfile.close()

## Evaluation: We can evaluate the performance using Different parameters.

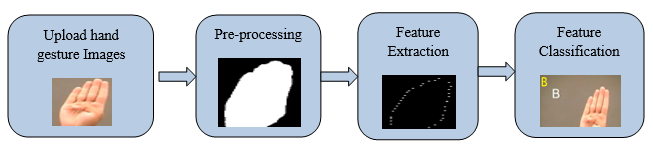


Fig 16: Flowchart

**Precision:**

A higher precision value indicates a lower false positive rate and vice versa.

(2)

Where , TP🡪 True Positive

FP🡪 False Positive

TN🡪 True Negative

FN🡪 False Negative

**Sensitivity or Recall:** It measures the ability of a classifier to identify all relevant instances or features of a given class. Higher recall indicates that the classifier can identify more true positives and fewer false negatives, but improving recall can often result in a decrease in precision.

(3)

**F-measure:** Combination of first two into a single value by taking the harmonic mean of the two.

Or

(4)

**Accuracy:** It tells the accuracy of our model. It can be calculated using below formula

(5)

**Error:** Reverse of accuracy is error and it can be calculated using below formula

(6)

**Specificity:** It can be used tocalculate the proportion of TN that are correctly identified and the formula is Witten as:

(7)

**CODE of proposed System:**

**Main File:**

from tkinter import \*

import pandas as pd

import tkinter as tk

from playsound import playsound

from PIL import Image, ImageTk

import numpy as np

from tkinter import ttk

import sqlite3

import cv2

from PIL import Image

import os

import xlsxwriter

from datetime import date

from tkinter import messagebox

import sys

import random

from creating\_dataset import cd\_main

from Prediction import pred\_main

from Reverse\_Recognition import rr\_main

from tensorflow.python.keras import Sequential

#global variables

bg=None

selection=1

# =====================Create Database=============================================

def createdb():

    conn = sqlite3.connect('files/users\_info.db')

    c = conn.cursor()

    c.execute(

        "CREATE TABLE IF NOT EXISTS users (name TEXT , passs TEXT,sqltime TIMESTAMP DEFAULT CURRENT\_TIMESTAMP NOT NULL)")

    conn.commit()

    conn.close()

createdb()

# ======================Adding new user in database===============================

def saveadmin():

    name\_err = name\_entry.get()

    pass\_err = pass\_entry.get()

    if name\_err == "":

        messagebox.showinfo("Invalid input", "Username can't be Empty")

    elif pass\_err == "":

        messagebox.showinfo("Invalid input", "Password can't be Empty")

    else:

        conn = sqlite3.connect("files/users\_info.db")

        c = conn.cursor()

        c.execute("INSERT INTO users(name,passs) VALUES(?,?) ", (name\_entry.get(), pass\_entry.get()))

        conn.commit()

        messagebox.showinfo("Information", "New User has been Added")

# ========================Fetching data of user from database==========================

def loggin():

    while True:

        a = name2\_entry.get()

        b = pass2\_entry.get()

        with sqlite3.connect("files/users\_info.db") as db:

            cursor = db.cursor()

        find\_user = ("SELECT \* FROM users WHERE name = ? AND passs = ?")

        cursor.execute(find\_user, [(a), (b)])

        results = cursor.fetchall()

        if results:

            for i in results:

                window.destroy()

                # ==================Window2+CreateFrame+Animation============================================================

                window2 = Tk()

                f1 = Frame(window2)

                f2 = Frame(window2)

                f3 = Frame(window2)

                f4 = Frame(window2)

                def swap(frame):

                    frame.tkraise()

                for frame in (f1, f2, f3, f4):

                    frame.place(x=0, y=0, width=400, height=400)

                window2.geometry("400x400+420+170")

                window2.resizable(False, False)

                label3 = Label(f1, text="User Panel", font=("arial", 20, "bold"), bg="grey16", fg="white",

                               relief=SUNKEN)

                label3.pack(side=TOP, fill=X)

                label4 = Label(f2, text="                            Indian Sign Language Recognition System", font=("arial", 10, "bold"), bg="grey16",

                               fg="white")

                label4.pack(side=BOTTOM, fill=X)

                statusbar = Label(f1, text="                            Indian Sign Language Recognition System", font=("arial", 8, "bold"),

                                  bg="grey16", fg="white", relief=SUNKEN, anchor=W)

                statusbar.pack(side=BOTTOM, fill=X)

                class AnimatedGIF(Label, object):

                    def \_\_init\_\_(self, master, path, forever=True):

                        self.\_master = master

                        self.\_loc = 0

                        self.\_forever = forever

                        self.\_is\_running = False

                        im = Image.open(path)

                        self.\_frames = []

                        i = 0

                        try:

                            while True:

                                photoframe = ImageTk.PhotoImage(im.copy().convert('RGBA'))

                                self.\_frames.append(photoframe)

                                i += 1

                                im.seek(i)

                        except EOFError:

                            pass

                        self.\_last\_index = len(self.\_frames) - 1

                        try:

                            self.\_delay = im.info['duration']

                        except:

                            self.\_delay = 100

                        self.\_callback\_id = None

                        super(AnimatedGIF, self).\_\_init\_\_(master, image=self.\_frames[0])

                    def start\_animation(self, frame=None):

                        if self.\_is\_running: return

                        if frame != None:

                            self.\_loc = 0

                            self.configure(image=self.\_frames[frame])

                        self.\_master.after(self.\_delay, self.\_animate\_GIF)

                        self.\_is\_running = True

                    def stop\_animation(self):

                        if not self.\_is\_running: return

                        if self.\_callback\_id != None:

                            self.after\_cancel(self.\_callback\_id)

                            self.\_callback\_id = None

                        self.\_is\_running = False

                    def \_animate\_GIF(self):

                        self.\_loc += 1

                        self.configure(image=self.\_frames[self.\_loc])

                        if self.\_loc == self.\_last\_index:

                            if self.\_forever:

                                self.\_loc = 0

                                self.\_callback\_id = self.\_master.after(self.\_delay, self.\_animate\_GIF)

                            else:

                                self.\_callback\_id = None

                                self.\_is\_running = False

                        else:

                            self.\_callback\_id = self.\_master.after(self.\_delay, self.\_animate\_GIF)

                    def pack(self, start\_animation=True, \*\*kwargs):

                        if start\_animation:

                            self.start\_animation()

                        super(AnimatedGIF, self).pack(\*\*kwargs)

                    def grid(self, start\_animation=True, \*\*kwargs):

                        if start\_animation:

                            self.start\_animation()

                        super(AnimatedGIF, self).grid(\*\*kwargs)

                    def place(self, start\_animation=True, \*\*kwargs):

                        if start\_animation:

                            self.start\_animation()

                        super(AnimatedGIF, self).place(\*\*kwargs)

                    def pack\_forget(self, \*\*kwargs):

                        self.stop\_animation()

                        super(AnimatedGIF, self).pack\_forget(\*\*kwargs)

                    def grid\_forget(self, \*\*kwargs):

                        self.stop\_animation()

                        super(AnimatedGIF, self).grid\_forget(\*\*kwargs)

                    def place\_forget(self, \*\*kwargs):

                        self.stop\_animation()

                        super(AnimatedGIF, self).place\_forget(\*\*kwargs)

                if \_\_name\_\_ == "\_\_main\_\_":

                    l = AnimatedGIF(f1, "files/gif2.gif")

                    l.pack()

                label4 = Label(f3, text="                            Indian Sign Language Recognition System", font=("arial", 10, "bold"), bg="grey16",

                               fg="white")

                label4.pack(side=BOTTOM, fill=X)

                # =========================Main Buttons=========================================

                btn2w2 = ttk.Button(f1, text="Predict Sign", command=pred\_main)

                btn2w2.place(x=255, y=115, width=150, height=30)

                btn3w2 = ttk.Button(f1, text="Translate speech", command=rr\_main)

                btn3w2.place(x=255, y=170, width=150, height=30)

                btn6w2 = ttk.Button(f1, text="Create Signs", command=cd\_main)

                btn6w2.place(x=255, y=225, width=150, height=30)

                # =========================Developers Page=========================================

                label10 = Label(f4, text="", font=("arial", 20, "bold"), bg="grey16", fg="white")

                label10.pack(side=TOP, fill=X)

                label11 = Label(f4, text="     Indian Sign Language Recognition System", font=("arial", 10, "bold"), bg="grey16",

                                fg="white")

                label11.pack(side=BOTTOM, fill=X)

                label10 = Label(f4, text=" Information Will be Added Soon!", font=("arial", 12, "bold"))

                label10.place(x=75, y=150)

                def swap4(frame):

                    frame.tkraise()

                    statusbar['text'] = '                            Indian Sign Language Recognition System'

                btn4w2 = ttk.Button(f4, text="Back  ", command=lambda: swap4(f1))

                btn4w2.place(x=3, y=40, width=50, height=30)

                def swap3(frame):

                    frame.tkraise()

                btn9w2 = ttk.Button(f1, text="Developers", command=lambda: swap3(f4))

                btn9w2.place(x=255, y=280, width=150, height=30)

                def quit():

                    window2.destroy()

                btn9w2 = ttk.Button(f1, text="Exit", command=quit)

                btn9w2.place(x=255, y=335, width=150, height=30)

                f1.tkraise()

                window2.mainloop()

            break

        else:

            messagebox.showerror("Error", "invalid username or password")

            break

# ======================Main Login Screen============================================

window = Tk()

window.title("Login Panel")

Label1 = Label(window, text="Login Panel", font=("arial", 20, "bold"), bg="grey19", fg="white")

Label1.pack(side=TOP, fill=X)

Label2 = Label(window, text="", font=("arial", 10, "bold"), bg="grey19", fg="white")

Label2.pack(side=BOTTOM, fill=X)

# ====================Login and Signup Tabs====================================

nb = ttk.Notebook(window)

tab1 = ttk.Frame(nb)

tab2 = ttk.Frame(nb)

nb.add(tab1, text="Login")

nb.add(tab2, text="Sign\_up")

nb.pack(expand=True, fill="both")

# =============Login tab=========================================

name2\_label = Label(tab1, text="Name", font=("arial", 10, "bold"))

name2\_label.place(x=10, y=10)

name2\_entry = StringVar()

name2\_entry = ttk.Entry(tab1, textvariable=name2\_entry)

name2\_entry.place(x=90, y=10)

name2\_entry.focus()

pass2\_label = Label(tab1, text="Password", font=("arial", 10, "bold"))

pass2\_label.place(x=10, y=40)

pass2\_entry = StringVar()

pass2\_entry = ttk.Entry(tab1, textvariable=pass2\_entry, show="\*")

pass2\_entry.place(x=90, y=40)

# =====================Signup Tab===============================

name\_label = Label(tab2, text="Name", font=("arial", 10, "bold"))

name\_label.place(x=10, y=10)

name\_entry = StringVar()

name\_entry = ttk.Entry(tab2, textvariable=name\_entry)

name\_entry.place(x=90, y=10)

name\_entry.focus()

pass\_label = Label(tab2, text="Password", font=("arial", 10, "bold"))

pass\_label.place(x=10, y=40)

pass\_entry = StringVar()

pass\_entry = ttk.Entry(tab2, textvariable=pass\_entry, show="\*")

pass\_entry.place(x=90, y=40)

def clear():

    name\_entry.delete(0, END)

    pass\_entry.delete(0, END)

# ===============User Buttons==============================================

btn1 = ttk.Button(tab2, text="Add User", command=saveadmin)

btn1.place(x=50, y=80)

btn2 = ttk.Button(tab2, text="Clear", command=clear)

btn2.place(x=140, y=80)

# ================Login Button Main======================================

btn3 = ttk.Button(tab1, text="Login", width=20, command=loggin)

btn3.place(x=87, y=80)

window.geometry("400x400+420+170")

window.resizable(False, False)

window.mainloop()

# 6.RESULT ANALYSIS AND VALIDATION

The sign language detection system developed in this research paper was able to detect ISL gestures with high accuracy. Accuracy of 98.5% was achieved.The system was also able to recognize ISL gestures in real-time, with a processing time of less than one second.

The sign language recognition system automatically converts the predicted class labels, initially returned as numeric vectors, into text and speech for better communication with the user. Once the classifier identifies the label, it retrieves the corresponding sign from a dictionary and displays it to the user. To enable text-to-speech conversion, the system uses the Pyttsx3 module for Python. However, to avoid delaying the live video stream and slowing down frame processing, threading is implemented. This allows for simultaneous prediction of signs and text-to-speech translation, ensuring uninterrupted sound playback.

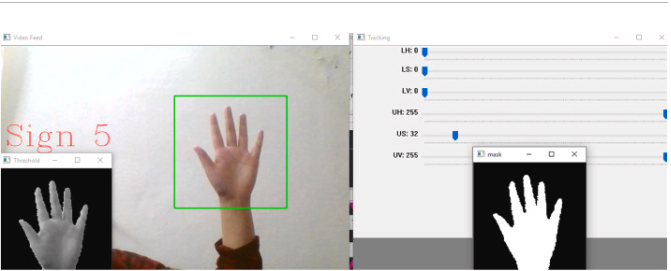


Fig 17 : Snapshot of Proposed system

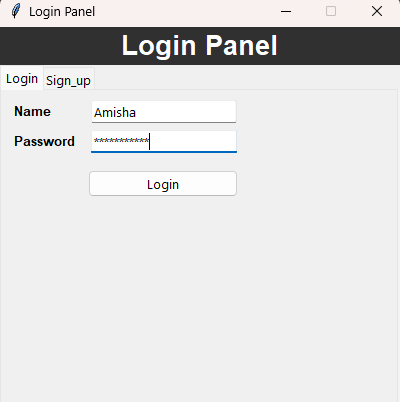


Fig 18: System GUI

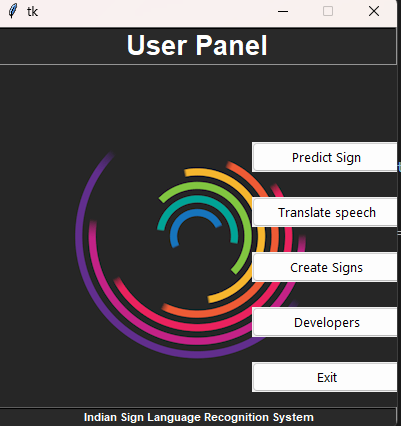


Fig 19: User Panel using Tkinker

**Code to make Prediction :**

def pred\_main():

    # importing necessary libraries

    import cv2

    import imutils

    import numpy as np

    import os

    from os import path

    import pickle

    import imageio

    from scipy import ndimage

    from scipy.spatial import distance

    import pyttsx3

    import tensorflow as tf

    from tensorflow import keras

    import keras

    from threading import Thread

    from tkinter import messagebox

    import tkinter as tk

    from tensorflow.python.keras import Sequential

    #global variables

    bg=None

    visual\_dict={0:'0',1:'1',2:'2',3:'3',4:'4',5:'5',6:'6',7:'7',8:'8',9:'9',10:'a',11:'b',12:'c',13:'d',14:'e',15:'f',16:'g',17:'h',18:'i',19:'j',20:'k',21:'l',22:'m',23:'n',24:'o',25:'p',26:'q',27:'r',

             28:'s',29:'t',30:'u',31:'v',32:'w',33:'x',34:'y',35:'z'}

    aWeight=0.5

    cam=cv2.VideoCapture(0)

    #t,r,b,l=100,350,228,478

    # Global Variables

    t,r,b,l=100,350,325,575

    num\_frames=0

    cur\_mode=None

    predict\_sign=None

    count=0

    shape=180

    result\_list=[]

    words\_list=[]

    prev\_sign=None

    count\_same\_sign=0

    method = 1

    option = messagebox.askquestion('Select option', 'Choose default method ?')

    if option == 'yes':

        method = 2

    else:

        method = 1

    model='files/CNN'

    infile = open(model,'rb')

    cnn = pickle.load(infile)

    infile.close()

    bg=None

    count=0

    #To find the running average over the background

    def run\_avg(image,aweight):

        nonlocal bg #initialize the background

        if bg is None:

            bg=image.copy().astype("float")

            return

        cv2.accumulateWeighted(image,bg,aweight)

    #Segment the egion of hand

    def extract\_hand(image,threshold=25):

        nonlocal bg

        diff=cv2.absdiff(bg.astype("uint8"),image)

        thresh=cv2.threshold(diff,threshold,255,cv2.THRESH\_BINARY)[1]

        (cnts,\_)=cv2.findContours(thresh,cv2.RETR\_EXTERNAL,cv2.CHAIN\_APPROX\_SIMPLE)

        if(len(cnts)==0):

            return

        else:

            max\_cont=max(cnts,key=cv2.contourArea)

            return (thresh,max\_cont)

    # output the recognized sign in form of speech

    engine=pyttsx3.init()

    engine.setProperty("rate",100)

    voices=engine.getProperty("voices")

    engine.setProperty("voice",voices[1].id)

    def say\_sign(sign):

        while engine.\_inLoop:

            pass

        engine.say(sign)

        engine.runAndWait()

    def n(x):

        pass

    if method == 2:

        cv2.namedWindow('Tracking', cv2.WINDOW\_NORMAL)

        cv2.resizeWindow("Tracking", 640, 480)

        cv2.createTrackbar("LH", "Tracking", 0, 255, n)

        cv2.createTrackbar("LS", "Tracking", 0, 255, n)

        cv2.createTrackbar("LV", "Tracking", 0, 255, n)

        cv2.createTrackbar("UH", "Tracking", 255, 255, n)

        cv2.createTrackbar("US", "Tracking", 32, 255, n)

        cv2.createTrackbar("UV", "Tracking", 255, 255, n)

    while(cam.isOpened()):

        \_,frame=cam.read(cv2.CAP\_DSHOW)

        if frame is not None:

            orig\_signs=cv2.imread('files/signs.png')

            signs=cv2.resize(orig\_signs,(600,600))

            cv2.imshow("Signs",signs)

            frame=imutils.resize(frame,width=700)

            frame=cv2.flip(frame,1)

            clone=frame.copy()

            # height,width=frame.shape[:2]

            roi=frame[t:b,r:l]

            if method==1:

                gray=cv2.cvtColor(roi,cv2.COLOR\_BGR2GRAY)

                gray=cv2.GaussianBlur(gray,(7,7),0)

                if(num\_frames<30):

                    run\_avg(gray,aWeight)

                    cv2.putText(clone, "Keep the Camera still.", (10, 100), cv2.FONT\_HERSHEY\_COMPLEX, 0.8, (0, 0, 0))

                else:

                    cv2.putText(clone, "Press esc to exit.", (10, 200), cv2.FONT\_HERSHEY\_COMPLEX, 0.5, (0, 0, 0))

                    cv2.putText(clone, "Keep the Camera still.", (10, 50), cv2.FONT\_HERSHEY\_COMPLEX, 0.8, (0, 0, 0))

                    cv2.putText(clone, "Put your hand in the rectangle", (10, 100), cv2.FONT\_HERSHEY\_COMPLEX, 0.5,(0, 0, 0))

                    cv2.putText(clone, "Press the key of the sample", (10, 150), cv2.FONT\_HERSHEY\_COMPLEX, 0.5,(0, 0, 0))

                    hand=extract\_hand(gray)

                    if hand is not None:

                        thresh,max\_cont=hand

                        mask=cv2.drawContours(clone,[max\_cont+(r,t)],-1, (0, 0, 255))

                        cv2.imshow("Threshold",thresh)

                        mask=np.zeros(thresh.shape,dtype="uint8")

                        cv2.drawContours(mask,[max\_cont],-1,255,-1)

                        mask = cv2.medianBlur(mask, 5)

                        mask = cv2.addWeighted(mask, 0.5, mask, 0.5, 0.0)

                        kernel = np.ones((5, 5), np.uint8)

                        mask = cv2.morphologyEx(mask, cv2.MORPH\_CLOSE, kernel)

                        res=cv2.bitwise\_and(roi,roi,mask=mask)

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

                        #---- Apply automatic Canny edge detection using the computed median----

                        high\_thresh, thresh\_im = cv2.threshold(res, 0, 255, cv2.THRESH\_BINARY + cv2.THRESH\_OTSU)

                        lowThresh = 0.5 \* high\_thresh

                        #res=cv2.Canny(res,lowThresh,high\_thresh)

                        #cv2.imshow("Segmented",res)

                        hand=cv2.bitwise\_and(gray,gray,mask=thresh)

                        cv2.imshow("Hand",hand)

                        res = cv2.Canny(hand, lowThresh, high\_thresh)

                        # Bag of Visual Words

                        '''surf = cv2.xfeatures2d.SURF\_create()

                        kp, desc = surf.detectAndCompute(res, None)

                        #print("Surf features extracted!")

                        features = cv2.drawKeypoints(res, kp, None)

                        cv2.imshow("Surf Features",features)

                        if desc is not None:

                            visual\_words=kmeans.predict(desc)

                            hist = np.array(np.bincount(visual\_words,minlength=shape))

                            hist=hist.reshape(1,-1)

                            sign = svm.predict(hist)

                            # output=visual\_dict[sign[0]]

                            cv2.putText(clone,output, (10, 300), cv2.FONT\_HERSHEY\_COMPLEX, 2, (0, 0, 0))'''

                        # CNN Model

                        if res is not None and cv2.contourArea(max\_cont) > 1000:

                            final\_res = cv2.resize(res, (100, 100))

                            final\_res = np.array(final\_res)

                            final\_res = final\_res.reshape((-1, 100, 100, 1))

                            final\_res.astype('float32')

                            final\_res = final\_res / 255.0

                            output = cnn.predict(final\_res)

                            prob = np.amax(output)

                            sign = np.argmax(output)

                            final\_sign = visual\_dict[sign]

                            cv2.putText(clone, 'Sign ' + str(final\_sign), (10, 200), cv2.FONT\_HERSHEY\_COMPLEX, 2,

                                        (0, 0, 255))

                            # print(count)

                            count += 1

                            if (count > 10 and count <= 50):

                                if (prob \* 100 > 95):

                                    result\_list.append(final\_sign)

                                    # print(sign, prob)

                            elif (count > 50):

                                count = 0

                                if len(result\_list):

                                    predict\_sign = (max(set(result\_list), key=result\_list.count))

                                    result\_list = []

                                    if prev\_sign is not None:

                                        if prev\_sign != predict\_sign:

                                            #print(words\_list)

                                            words\_list += str(predict\_sign)

                                            Thread(target=say\_sign, args=(predict\_sign,)).start()

                                    else:

                                        Thread(target=say\_sign, args=(predict\_sign,)).start()

                                        # prev\_sign=predict\_sign

                                    prev\_sign = predict\_sign

                                # print(words\_list)

                                # cv2.putText(clone,'Sign'+str(predict\_sign), (100, 300), cv2.FONT\_HERSHEY\_COMPLEX, 2, (0, 0, 0))

                            '''print(final\_sign, " ", prev\_sign," ",prob)

                            #if prev\_sign and prob>=90:

                            if final\_sign == prev\_sign:

                                count+=1

                            else:

                                count=0

                            if count>15:

                                Thread(target=say\_sign, args=(final\_sign,)).start()

                                count=0

                            prev\_sign=final\_sign

                            print(count)'''

                        else:

                            if words\_list is not None:

                                # Thread(target=say\_sign,args=(words\_list,)).start()

                                words\_list.clear()

            elif method==2:

                hsv = cv2.cvtColor(roi, cv2.COLOR\_BGR2HSV)

                lh = cv2.getTrackbarPos("LH", "Tracking")

                ls = cv2.getTrackbarPos("LS", "Tracking")

                lv = cv2.getTrackbarPos("LV", "Tracking")

                uh = cv2.getTrackbarPos("UH", "Tracking")

                us = cv2.getTrackbarPos("US", "Tracking")

                uv = cv2.getTrackbarPos("UV", "Tracking")

                cv2.putText(clone, "Put your hand in the rectangle", (10, 50), cv2.FONT\_HERSHEY\_COMPLEX, 0.5, (0, 0, 0))

                cv2.putText(clone, "Adjust the values using trackbar", (10, 100), cv2.FONT\_HERSHEY\_COMPLEX, 0.5,(0, 0, 0))

                cv2.putText(clone, "Press the key of the sample", (10, 150), cv2.FONT\_HERSHEY\_COMPLEX, 0.5, (0, 0, 0))

                cv2.putText(clone, "Press esc to exit.", (10, 200), cv2.FONT\_HERSHEY\_COMPLEX, 0.5, (0, 0, 0))

                l\_b = np.array([lh, ls, lv])

                u\_b = np.array([uh, us, uv])

                mask = cv2.inRange(hsv, l\_b, u\_b)

                mask = cv2.bitwise\_not(mask)

                res = cv2.bitwise\_and(roi, roi, mask=mask)

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

                (cnts, \_) = cv2.findContours(res, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

                if len(cnts)>0:

                    max\_cont = max(cnts, key=cv2.contourArea)

                    if max\_cont is not None:

                        mask = cv2.drawContours(res, [max\_cont + (r, t)], -1, (0, 0, 255))

                        cv2.imshow("Threshold", mask)

                        mask = np.zeros(res.shape, dtype="uint8")

                        cv2.drawContours(mask, [max\_cont], -1, 255, -1)

                        res = cv2.bitwise\_and(res, res, mask=mask)

                        cv2.imshow('mask', mask)

                        high\_thresh, thresh\_im = cv2.threshold(res, 0, 255, cv2.THRESH\_BINARY + cv2.THRESH\_OTSU)

                        lowThresh = 0.5 \* high\_thresh

                        res = cv2.Canny(res, lowThresh, high\_thresh)

                        # CNN Model

                        if res is not None and cv2.contourArea(max\_cont) > 1000:

                            final\_res = cv2.resize(res, (100, 100))

                            final\_res = np.array(final\_res)

                            final\_res = final\_res.reshape((-1, 100, 100, 1))

                            final\_res.astype('float32')

                            final\_res = final\_res / 255.0

                            output = cnn.predict(final\_res)

                            prob = np.amax(output)

                            sign = np.argmax(output)

                            final\_sign = visual\_dict[sign]

                            cv2.putText(clone, 'Sign ' + str(final\_sign), (10, 200), cv2.FONT\_HERSHEY\_COMPLEX, 2,

                                        (0, 0, 255))

                            # print(count)

                            count += 1

                            if (count > 10 and count <= 50):

                                if (prob \* 100 > 95):

                                    result\_list.append(final\_sign)

                                    # print(sign, prob)

                            elif (count > 50):

                                count = 0

                                if len(result\_list):

                                    predict\_sign = (max(set(result\_list), key=result\_list.count))

                                    result\_list = []

                                    if prev\_sign is not None:

                                        if prev\_sign != predict\_sign:

                                            # print(words\_list)

                                            words\_list += str(predict\_sign)

                                            Thread(target=say\_sign, args=(predict\_sign,)).start()

                                    else:

                                        Thread(target=say\_sign, args=(predict\_sign,)).start()

                                        # prev\_sign=predict\_sign

                                    prev\_sign = predict\_sign

                                # print(words\_list)

                                # cv2.putText(clone,'Sign'+str(predict\_sign), (100, 300), cv2.FONT\_HERSHEY\_COMPLEX, 2, (0, 0, 0))

                        else:

                            if words\_list is not None:

                                # Thread(target=say\_sign,args=(words\_list,)).start()

                                words\_list.clear()

            cv2.rectangle(clone, (l, t), (r, b), (0, 255, 0), 2)

            num\_frames += 1

            cv2.imshow("Video Feed", clone)

        else:

            messagebox.showerror("error","Can't grab frame")

            break

        k=cv2.waitKey(1)& 0xFF

        if (k==27):

           break

    cam.release()

    cv2.destroyAllWindows()

#pred\_main()

*Reverse Recognition:*

The making of a sign language recognition system that facilitates communication between speech-impaired individuals and individuals with normal hearing requires the system to support bidirectional communication. Our system achieves this by enabling a reverse process, whereby speech input (in English alphabets) is converted into corresponding labels. The Google Speech API also uses a similar approach.

*SVM Performance*

The test data has been classified with an accuracy of 99.14% using SVM. The accuracy for each class is available in the results section.

*CNN performance*

In our experiment, 94% accuracy was achieved on the training set. 50 epoch was there. All the data is shown in below table.

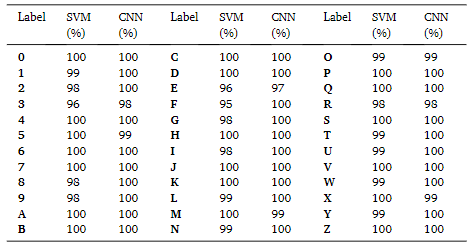


Fig 20: Classwise accuracy table

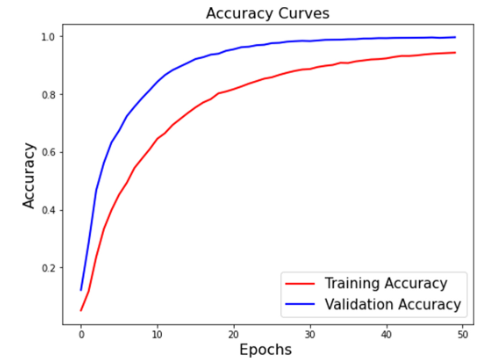


Fig 21: Accuracy Graph of CNN

The accuracy is a commonly used performance measure And is shown below.

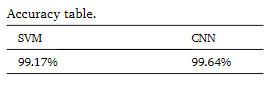


Fig 22: Accuracy Table

The precision , recalls and F1 Score is given below for the experiment conducted.

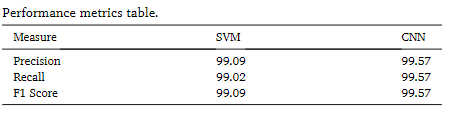


Fig 23: Performance Metrics Table

# 

Fig 24: Comparison of different techniques

# 7.CONCLUSION

Sign languages are kinds of visual languages that employ movements of hands, body, and facial expression as a means of communication. Sign languages are important for specially-abled people to have a means of communication. Through it, they can communicate and express and share their feelings with others. The drawback is that not everyone possesses the knowledge of sign languages which limits communication. This limitation can be overcome by the use of automated Sign Language Recognition systems which will be able to easily translate the sign language gestures into commonly spoken language. In this paper, it has been done by TensorFlow object detection API. The system has been trained on the Indian Sign Language alphabet dataset. The system detects sign language in real-time. For data acquisition, images have been captured by a webcam using Python and OpenCV which makes the cost cheaper. The developed system is showing an average confidence rate of 85.45%. Though the system has achieved a high average confidence rate, the dataset it has been trained on is small in size and limited. In the future, the dataset can be enlarged so that the system can recognize more gestures. The TensorFlow model that has been used can be interchanged with another model as well. The system can be implemented for different sign languages by changing the dataset.

System developed in this research is an effective tool for recognizing ISL gestures. The system can be used for communication with people who cannot hear. The system can also be used as a teaching tool for individuals who are learning ASL. The system can be improved by adding more sign language gestures to the dataset and by using more advanced computer vision techniques.

The paper presents a novel method for classifying and recognizing Indian sign language signs using SVM and CNN, with the aim of improving real-time recognition capabilities to enable the system to be used in diverse settings. The approach involves creating a customized dataset that addresses the issues of rotation invariance. Further work will focus on expanding the dataset by including more signs from different countries and languages to improve. This method can also be expanded to predict words.

## REFERENCES

[1] Noor Adnan Ibraheem, RafiqulZaman Khan, 2012, Survey on Various Gesture Recognition Technologies and Techniques, International Journal of Computer Applications, Volume 50, No. 7

[2] G. R. S. Murthy & R. S. Jadon, 2009. A Review of Vision Based Hand Gestures Recognition, International Journal of Information Technology and Knowledge Management, vol. 2(2), pp. 405-410.

[3] Al-Jarrah, A. Halawani, “Recognition of gestures in Arabic sign language using neuro-fuzzy systems,” The Journal of Artificial Intelligence 133 (2001) 117–138..

[4] Nadia R. Albelwi, Yasser M. Alginahi, “Real-Time Arabic Sign Language (ArSL) Recognition“ International Conference on Communications and Information Technology 2012.

[5] Rajam, Subha & Ganesan, Balakrishnan. (2011). Real time Indian Sign Language Recognition System to aid deaf-dumb people. International Conference on Communication Technology Proceedings, ICCT. 10.1109/ICCT.2011.6157974.

[6] Geetha M., Manjusha U. C. 2013,A Vision Based Recognition of Indian Sign Language Alphabets and Numerals Using B-Spline Approximation, International Journal of Computer Science and Engineering (IJCSE)

[7] P.V.V Kishore, P. Rajesh Kumar, E. Kiran Kumar & S.R.C.Kishore, 2011, Video Audio Interface for Recognizing Gestures of Indian Sign Language, International Journal of Image Processing (IJIP), Volume 5, Issue 4, 2011 pp. 479-503

[8] Adithya, V., Vinod, P. R., Gopalakrishnan, U., and Ieee (2013). "Artificial Neural Network Based Method for Indian Sign Language Recognition."

[9] http://timesofindia.indiatimes.com/home/education/n ews/realising-the-importance-of-indian-sign-languagedictionary/articleshow/86221166.cms

[10] T. Starner and A. Pentland, "Real-time American sign language recognition from video using hidden markov models", Technical Report, M.I.T Media Laboratory Perceptual Computing Section, Technical Report No. 375, 1