SIGN LANGUAGE CONVERSION TO AUDIO WITH SUBTITLE GENERATION FOR SENSORY IMPAIRED PEOPLE USING COMPUTER VISION

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

By

A.S.P.MANIKANTA (39110026)
A.VINAY KUMAR (39110056)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SCHOOL OF COMPUTING

SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

Accredited with Grade "A" by NAAC | 12B Status by UGC | Approved by AICTE
JEPPIAAR NAGAR, RAJIV GANDHI SALAI,

CHENNAI - 600119

APRIL - 2023

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited with Grade "A" by NAAC | 12B Status by UGC | Approved by AICTE

www.sathyabama.ac.in

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the Bonafide work of **A.S.P.MANIKANTA (Reg.No - 39110026) and A.VINAY KUMAR (Reg.No - 39110056)** who carried out the Project entitled **"SIGN LANGUAGE CONVERSION WITH SUBTITLE GENERATION FOR SENSORY IMPAIRED PEOPLE USING COMPUTER VISION"** under my supervision from January 2023 to April 2023.

[illegible]

Internal Guide
Dr. B.U.ANU BARATHI, M.E., Ph.D.

[illegible]

SATHYABAMA INSTITUTE OF SCIENCE AND TECHNOLOGY
DEPARTMENT OF CSE

Head of the Department

Dr. L. LAKSHMANAN, M.E., Ph.D.

Submitted for Viva Voce Examination held on 26.04.2023

[illegible]

Internal Examiner

[illegible]

External Examiner

DECLARATION

I, **A.S.P.MANIKANTA (Reg.No- 39110026),** hereby declare that the Project Report entitled **“SIGN LANGUAGE CONVERSION WITH SUBTITLE GENERATION FOR SENSORY IMPAIRED PEOPLE USING COMPUTER VISION”** done by me under the guidance of **Dr. B.U.ANU BARATHI, M.E., Ph.D**. is submitted in partial fulfilment of the requirements for the award of Bachelor of Engineering degree in **Computer Science and Engineering**.

[illegible]

DATE: 26/04/2023

PLACE: Chennai **SIGNATURE OF THE CANDIDATE**

ACKNOWLEDGEMENT

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

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

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

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

ABSTRACT

Humans beings communicate and interact with each other to share their ideas and feelings but in case of sensory impaired people they can’t communicate just like normal human beings. So, sign language came into the picture and act as a medium for communication for sensory impaired people. The aim of this work is to develop a system for recognizing sign Language and converting it into audio. In the current work we use computer vision (cv) for detection of hand gestures, Decision Tree machine leaning algorithm for classification and text-to-speech API for converting text to corresponding speech. Recognition is a breakthrough for helping deaf-mute people and has been researched for many years. Unfortunately, every research has its own limitations and are still unable to be used commercially. Some of the researches have known to be successful but cheap method is needed for the Sign Language Recognition System to be commercialized..

TABLE OF THE CONTENT

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

Typically, sign or signed languages are employed to communicate meaning visually. These dialects are conveyed using non-manual features in addition to physical articulations. These are completely natural languages with their own lexicon and vocabulary. Smart methods for hand motion detection from photographs in hand signals are crucial in this situation. Many people throughout the world who have speech disorders and loss of hearing can benefit from these clever techniques. For accurately recognizing symbols and signs, various approaches have indeed been documented in the literature.

Innovations like movement and face recognition have gained significant traction in the basic sign language field in recent years. Different movements known as gestures are utilized during communicating. Hand or body movements are used. Gestures used in sign language typically involve visually communicated patterns. There are roughly 4,94,93,500.000 people with hearing impairments worldwide. Some of the existing methods for translating sign language take into account hand position, hand size, and finger movements.

Each sign in sign language has a particular meaning ascribed to it so that people can easily comprehend and interpret it. The people are creating distinct and unique sign languages depending on their native tongues and geographic locations. As spoken language, there are several variations in sign language. Around the world, more than 300 different sign languages are in use. They differ from one country to the next.

Sign language can have many diverse regional dialects that cause subtle variances in how people use and comprehend signs, even in nations where the same language is spoken. There are significant variances between some of the most popular sign languages, even though there are some commonalities. Additionally, not just the symptoms differ.

There are numerous varieties of sign language used not only in the UK but all over the world since the speaker's nonverbal cues, actions, and visual emotions may all have a huge impact on how a sign language is transmitted. Similar to spoken language, various groups and cultures create their own means of communication specific to the environment in which they exist.

For instance, native English speakers can be found in both Britain and the United States. However, there are considerable differences between American and British sign languages. Here is where a lot of companies and institutions still have trouble reaching out to the Deaf and Hard of Hearing people. Here are a few different types of sign language as mentioned below:

Indian Sign Language

Indian Sign Language uses both the right and left hands to depict a variety of hand gestures. Indian Sign Language serves as both a communication tool for the deaf and a representation of their pride and identity. The Rights of Persons with Disabilities (RPwD) Act 2016, which took effect on April 19, 2017, acknowledges sign language as a form of interaction that is particularly helpful when speaking with people who have hearing loss.

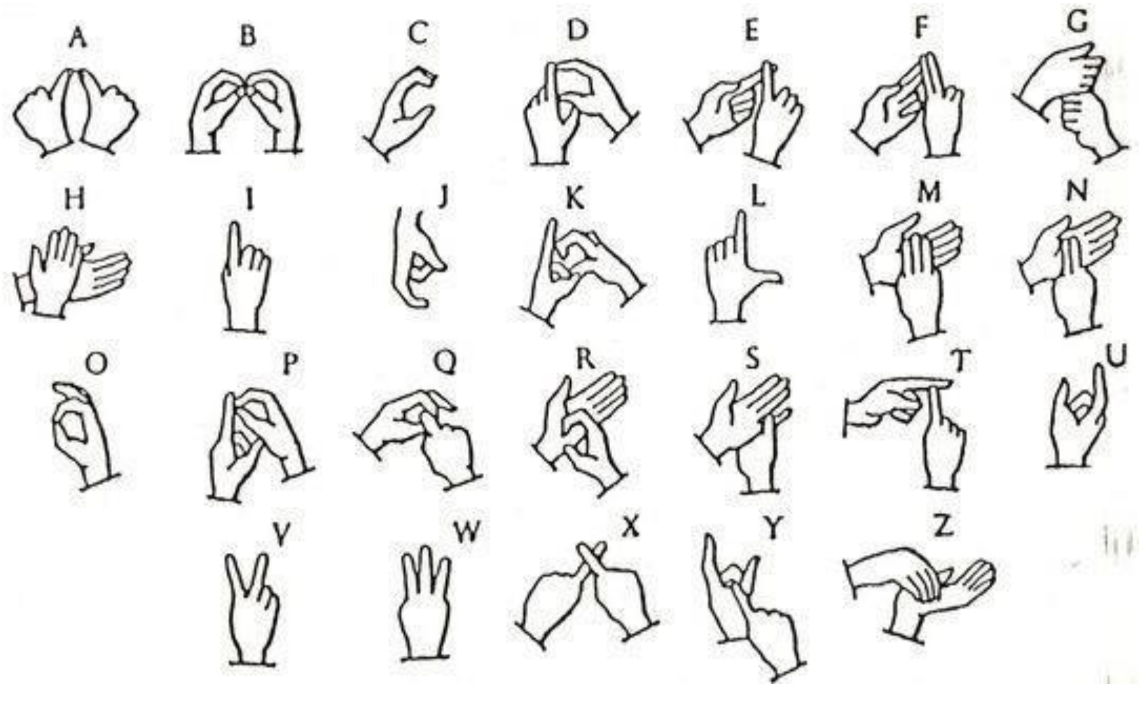


Fig.1.1 Indian Sign Language

Authorities are also required under the Act to encourage the use of sign language so that people with hearing loss can take role in society and benefit to it. The ISL gradually changed from 4000 words to 10,000 words and changed its goal of being used more widely.

American Sign Language (ASL)

On the other hand, one hand is all that's required to sign in ASL. As a result, putting the system into place is made simple. ASL has its own growth path and is independent of all spoken languages. The main sign language used by Deaf people in the United States and most of Anglophone Canada is American Sign Language (ASL), a natural language. ASL is a comprehensive and well-structured visual language that uses both manual and non-manual characteristics to communicate. Dialects of ASL and creoles based on ASL are spoken around the world, including much of West Africa and portions of Southeast Asia, besides North America. As a lingua franca, ASL is also commonly studied as a second or foreign language. French Sign Language is most closely similar to ASL. Although ASL exhibits characteristics untypical of creole languages, like agglutinative morphology, it has been argued that ASL is a creole language of the FSL.

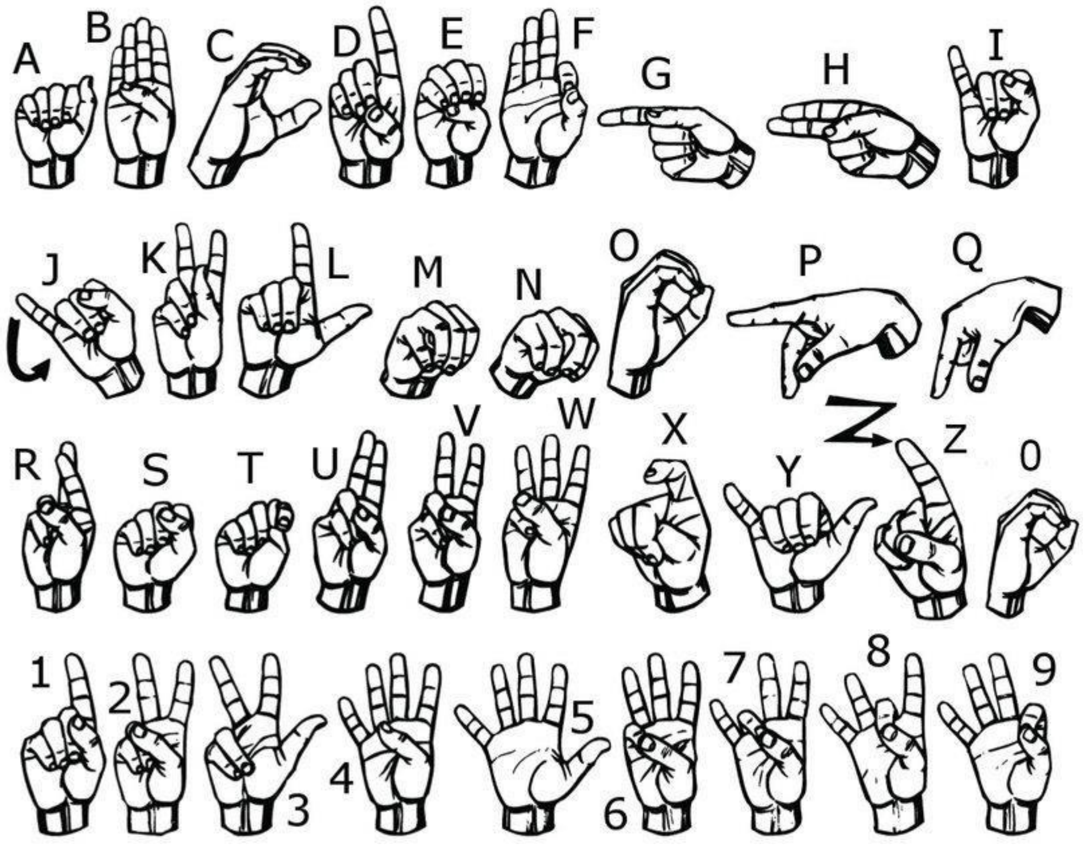


Fig 1.2 American Sign Language

French Sign Language (FSL)

The gesture used by the deaf in France and the French-speaking regions of Switzerland is known as French Sign Language (LSF). Ethnologies estimates that 100,000 native signers are involved. Dutch Sign Language (NGT), Flemish Sign Language (VGT), Belgian-French Sign Language (LSFB), Irish Sign Language (ISL), American Sign Language (ASL), Quebec (also known as French Canadian) Sign Language (LSQ), Brazilian Sign Language (LSB, LGB or LSCB), and Russian Sign Language are all linked to and partly descended from French Sign Language (RSL).

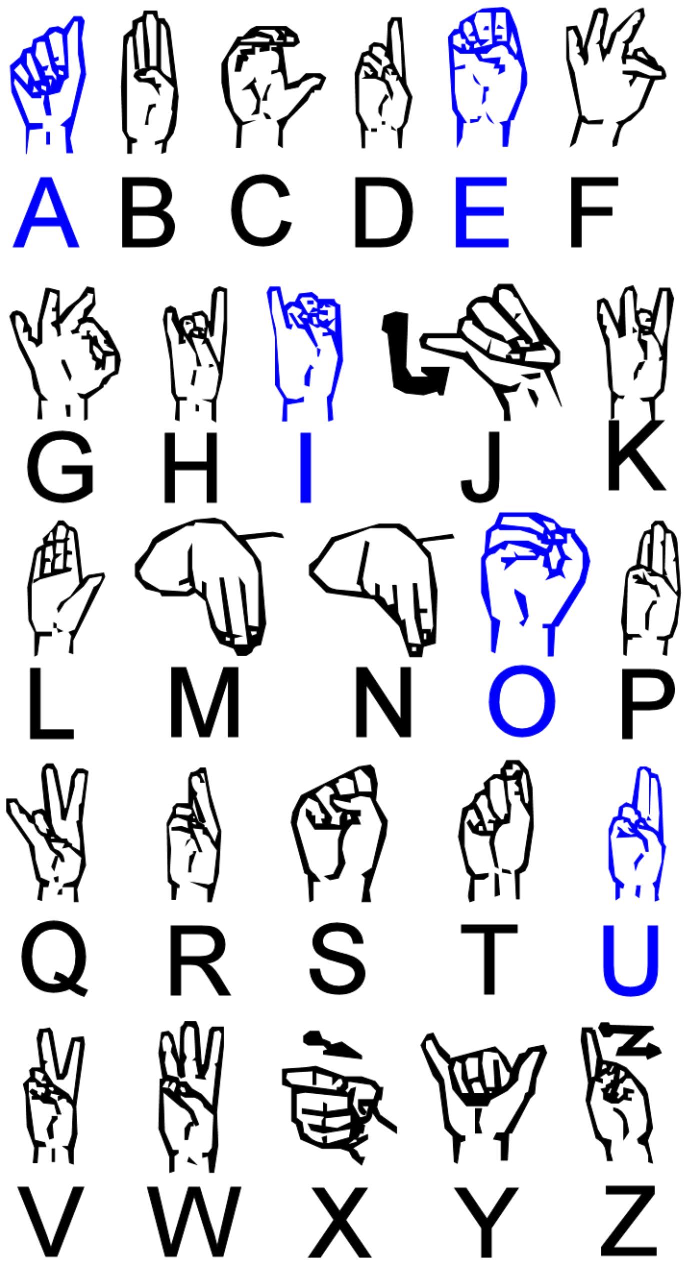


Fig 1.3 French Sign Language

The invention of French Sign Language is usually, though incorrectly, credited to Charles Michel de l'Épée (abbé de l'Épée). A pair of deaf twin sisters were inside a nearby house when he entered, seeking shelter from the rain. He was immediately struck by the depth and sophistication of the language they were using to communicate with one another and the deaf society in Paris. In fact, he is said to have discovered the language by complete accident. The abbé began studying what is now known as Old French Sign Language, and eventually he founded a free education for the hearing impaired. He created a methodology at this school to instruct his children how to read and write that he dubbed "methodical symbols."

British Sign Language (BSL)

The Deaf community in the UK uses the sign language known as British Sign Language (BSL), which is their first or chosen language. The British Deaf Association calculates that there are 151,000 BSL users in the UK, of whom 87,000 are Deaf, based on the percentage of people who reported "using British Sign Language at home" on the 2011 Scottish Census.

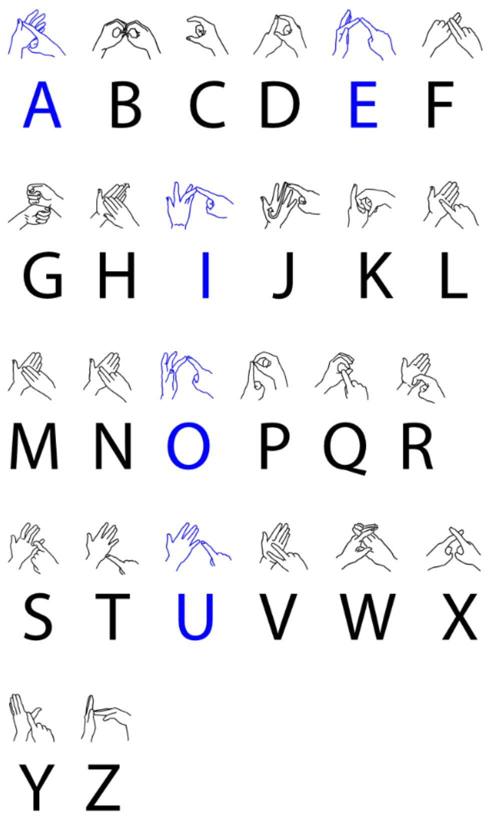


Fig 1.4 British Sign Language

In comparison, 15,000 persons who resided in England and Wales identified using BSL as their primary language in the 2011 England and Wales Census. As having to hear relatives of the deaf, basic sign translators, or as a consequence of other interactions with the British Deaf community, individuals who aren't deaf might also use BSL. The movement of the hands, body, face, and head as well as the utilization of space are all part of the language

Elements Of Sign Language

SL is a visual-spatial language based on positional and visual cues, including arm and body movements, the position and orientation of the hands, and the shape of the fingers and hands. Together, these elements are employed to communicate an idea's meaning. There are typically five components to the phonological structure of SL. In Second Life, each motion is made up of five different components. These five building pieces serve as SL's most valuable components, and automated, intelligent systems can use them to recognize SL.

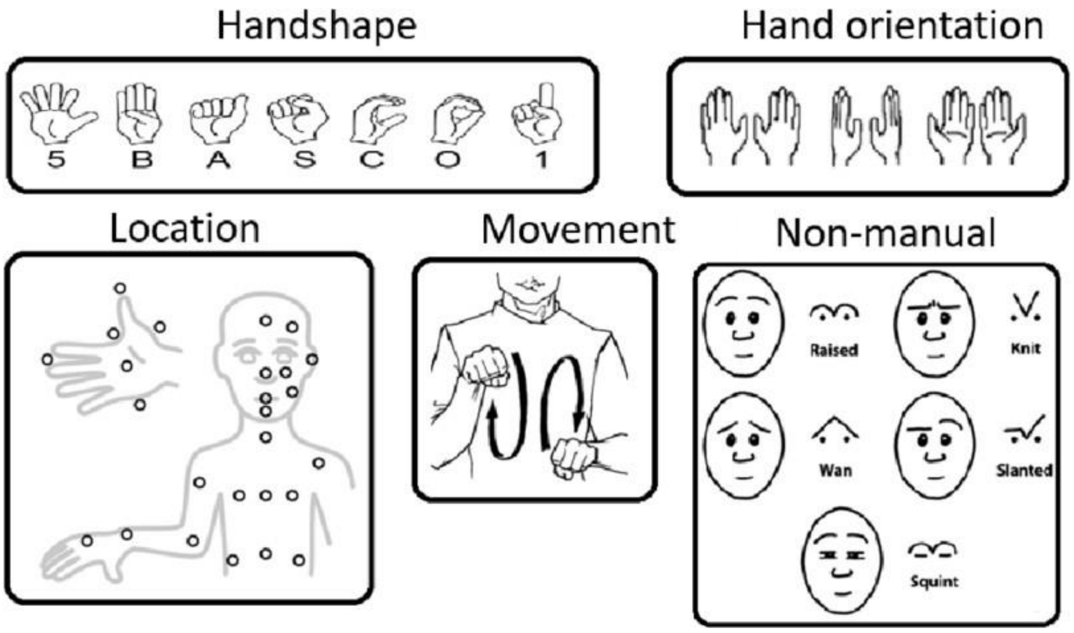


Fig 1.5 Elements Of Sign Language

Methods Of Sign Language Detection

Scholarly approaches to overcoming challenges brought on by disabilities are numerous, methodical, and context-specific. SLR systems, which are used to translate the signals of SL into text or speech to establish communication to individuals who do not understand these signs, are one of the crucial interventions. One of the most important projects aimed at collecting data for the motion of human hands is the development of SLR systems based on the sensory glove. To record hand layouts and identify the related meanings of gestures, three strategies are used: vision-based, sensor-based, and a mix of the two

Vision-Based Strategies:

Lenses are the main equipment used by vision-based systems to gather the required input data. The primary benefit of employing a camera is that it eliminates the requirement for sensors in sensory gloves and lowers the system's construction expenses. Due to the blur produced by a web camera, most laptops employ a highend camera, which is quite inexpensive. Despite the high-quality cameras that are found in most smartphones, there are still a number of issues that hinder the creation of real-time recognition applications. These issues include the limited field of view of the capturing device, high computational costs, and the requirement for multiple cameras in order to capture robust results (due to an issue of complexity and obstruction).

Sensory-Based Strategies

An alternate method for gathering information on gestures is to utilize a certain kind of instrumented glove that is equipped with flexion (or bend) sensors, accelerometers (ACCs), proximity sensors, and abduction sensors. The bend angles of the fingers, the abduction between the fingers, and the wrist's orientation (roll, pitch, and yaw) are all measured using these sensors. Depending on the number of sensors incorporated into the glove, different degrees of freedom (DoF) can be achieved using such gloves, ranging from 5 to 22. Glove-based systems have a significant benefit oversight systems in that they may immediately report necessary and pertinent information (degree of bend, pitch, etc.) to the portable computer in the form of voltage values obviating the need to transform raw data into useful values.

CHAPTER 2

LITERATURE SURVEY

2.1 INFERENCES FROM LITREATURE SURVEY

[1] Soma Shrenika, Myneni Madhu Bala, “Sign Language Recognition Using Template Matching Technique”, International Conference on Computer Science, Engineering and Applications (ICCSEA), 2020 Normal people cannot understand the signs used by the deaf because they do not understand the meaning of each sign. This is the goal of the system proposed here. This system employs a camera to record various hand gestures. Following that, the image is processed using various algorithms. Initially, the image is pre-processed. The edges are then determined using an edge detection algorithm. Finally, the sign is identified and the text is displayed using a template-matching algorithm. Because the output is text, the meaning of a specific sign can be easily interpreted. The system is implemented in Python using OpenCV.

[2] H Muthu Mariappan, V Gomathi, “Real-Time Recognition of Indian Sign Language”, International Conference on Computational Intelligence in Data Science (ICCIDS), 2019 The skin segmentation feature of OpenCV is used to identify and track the Regions of Interest (ROI) for recognizing the signs. The fuzzy c-means clustering machine learning algorithm is used to train and predict hand gestures. Many applications for gesture recognition exist, including gesture controlled robots and automated homes, game control, Human-Computer Interaction (HCI), and sign language interpretation. The proposed system can recognize real-time signs. As a result, it is extremely beneficial for hearing and speech impaired people to communicate with normal people.

[3] Wanbo Li, Hang Pu, Ruijuan Wang, “Sign Language Recognition Based on Computer Vision”, IEEE International Conference on Artificial Intelligence and Computer Applications (ICAICA), 2021 This system makes use of a PyQt-designed GUI interface. Users can choose between sign language recognition and translation, image capture with OpenCV, and special processing with the trained CNN neural network. Through LSTM decisions, the model can then identify American sign language. The user can also use the voice button to have the system convert the corresponding gesture image into the same pixels and write it to the video file based on the user's voice. The experimental results show that the rate of sign language recognition is 95.52% when compared to similar algorithms [5,

6], and the rate of sign language [7] (American sign language and Arabic numerals) is 90.3%.

[4] Pushkar Kurhekar, Janvi Phadtare, Sourav Sinha, Kavita P. Shirsat, “Real Time Sign Language Estimation System”, 3rd International Conference on Trends in Electronics and Informatics (ICOEI), 2019 This paper describes a system that allows people with speech and hearing impairments to communicate freely. The model can extract signs from videos by processing them frame by frame in a minimally cluttered environment. This sign is then displayed in text form. For webcam input and displaying the predicted sign, the system employs a Convolutional Neural Network (CNN) and fastai - a deep learning library - in conjunction with OpenCV. The experimental results show good sign segmentation in various backgrounds and high accuracy in gesture recognition.

[5] Aniket Kumar, Mehul Madaan, Shubham Kumar, Aniket Saha, Suman Yadav, “Indian Sign Language Gesture Recognition in Real-Time using Convolutional Neural Networks”, 8th International Conference on Signal Processing and Integrated Networks (SPIN), 2021 Using Convolutional Neural Networks, this paper proposes a model for identifying and classifying Indian Sign Language gestures in real-time (CNN). The model, which was built with OpenCV and Keras CNN implementations, aims to classify 36 ISL gestures representing 0-9 numbers and A-Z alphabets by transforming them to text equivalents. The dataset that was created and used consists of 300 images for each gesture that were fed into the CNN model for training and testing. The proposed model was implemented successfully and achieved 99.91% accuracy for the test images.

[6] Sai Nikhilesh Reddy Karna, Jai Surya Kode, Suneel Nadipalli, Sudha Yadav, “American Sign Language Static Gesture Recognition using Deep Learning and Computer Vision”, 2nd International Conference on Smart Electronics and Communication (ICOSEC), 2021 This study proposes a real-time hand-gesture recognition system based on the American Sign Language (ASL) dataset, with data captured using a BGR webcam and processed using Computer Vision (OpenCV). The Vision Transformer Model was used to train the 29 static gestures (the alphabet) from the official, standard ASL dataset (ViT). After getting trained with 87,000 RGB samples for 1 epoch, the model had an accuracy rate of 99.99%. (2719 batches of 32 images each).

[7] Shirin Sultana Shanta, Saif Taifur Anwar, Md. Rayhanul Kabir, “Bangla Sign Language Detection Using SIFT and CNN”, 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT), 2018.There have been very few studies on detecting Bangla sign language. The classifiers used by the majority of researchers in this field were SVM, ANN, or KNN. In this paper, the researchers attempt to develop a Bangla sign language system that uses SIFT feature extraction and Convolutional Neural Network (CNN) classification. The researchers also show that using the SIFT feature improves CNN's detection of Bangla Sign language.

[8] Rajiv Ranjan, B Shivalal Patro, Mohammad Daim Khan, Manas Chandan Behera, Raushan Kumar, Utsav Raj, “A Review on Sign Language Recognition Systems”, IEEE 2nd International Conference on Applied Electromagnetics, Signal Processing, & Communication (AESPC), 2022. For the enhancement of the recognized gesture, open CV, Gaussian blur, and contour approximation techniques are used. The TensorFlow framework is best suited for accurate software prediction with a CNN model of VGG-16 architecture and the Transfer Learning training method. The use of these tools results in a more accurate prediction of sign language gestures in the English alphabet. Given the project's future scope, the technology stack is both robust and scalable. This review paper discusses various methods and structures for sign language recognition.

[9] Kohsheen Tiku, Jayshree Maloo, Aishwarya Ramesh, Indra R, “Real-time Conversion of Sign Language to Text and Speech”, Second International Conference on Inventive Research in Computing Applications (ICIRCA), 2020.
The purpose of this paper is to examine the performance of various techniques for converting sign language to text/speech format. Following analysis, an Android application is created that can convert real-time ASL (American Sign Language) signs to text/speech.

[10] Vishwa Hariharan Iyer, U.M Prakash, Aashrut Vijay, P. Sathishkumar, “Sign Language Detection using Action Recognition”, 2nd International Conference on Advance Computing and Innovative Technologies in Engineering (ICACITE), 2022 This proposal is centered on the continuous detection of image frames in real-time using action detection to detect the user's action. After recognising key points using mediapipe holistic, which includes face, pose, and hand features, the model employs an LSTM neural network model. Collecting key value points for training and testing, pre-processing the data, and creating labels and features are all part of the proposed work.

Related works that was done have limitations as well as scope to be developed. The main limitations of these paper, there was no combined guidelines to help a visually impaired people and deaf people to communicate in sign language effectively. Our proposed system is dealt with to solve the limitations of these papers and providing a proper guideline also always forewarn from the obstacle faced simultaneously.

2.2 OPEN PROBLEM IN EXISTING SYSTEM

Existing System

In face of the problems often found with other learning models, such as the presence of multiple local minima and the curse of dimensionality, existing methods have been significantly ineffective therefore. The SVM method does not have any edge over the curse of dimensionality, thus their practical importance can be diminished. An immediate problem arising from the SVM's original hyperplane formulation is that it is not very obvious how to make the model applicable to more than two classes. Several approaches have been proposed to overcome this limitation, two of them being known as the 1-vs-1 and 1-vs-all strategies for multiple class classification. For a decision problem over classes, 1-vs-all requires the creation of classifiers, each trained to distinguish one class from the others. The decision then is taken in a winner-takes-all approach. However there is no clear indication this approach results in an optimum decision. As a result, we have proposed a model that solves these problems and offers higher accuracy and reliability as compared to traditional classifiers method.

Existing System

• Previous literature in this topic has in particular focused on precise sign language technologies which includes video and sensor-based totally signal language reputation and language translation. • Sign Language Recognition (SLR) includes the improvement of effective gadget mastering algorithms to reliably discover human articulation including unmarried characters or non-stop sentences.

• Limitations of present day SLR are associated with the shortage of big-scale observed records, which have an effect on the accuracy and generality of SLR techniques and the issue of defining man or woman limits in non-stop SLR scenarios.

• Sign Language Translation (SLT) interpretation between distinctive sign languages, as well as interpretation among sign and spoken languages.

• Difficulties in SLT rise up from the lack of information for multilingual sign language, as well as inaccuracies in SLR methods.

• The proposed system aims to introduce deep learning algorithms to resolve issues that rise up within the present machine.

• Difficulties in SLT stand up from the dearth of information for multilingual sign language, in addition to inaccuracies in SLR techniques.

• The proposed machine amis to introduce deep learning algorithms to solve problems that arise in the existing computer systems.

Disadvantages of the Existing System

• Only images are diagnosed

• Minimum accuracy 83%

• This is a big problem for CNN developers.

• An green device does no longer arise with CV Open

CHAPTER 3
REQUIREMENT ANALYSIS

3.1 FEASIBILITY STUDIES/RISK ANALYSIS OF THE PROJECT

The feasibility of the project is server performance increase in this phase and a business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis, the feasibility study of the proposed system is to be carried out. For feasibility analysis, some understanding of the major requirements for the system is essential. Three key considerations involved in the feasibility analysis are

- Economical feasibility
- Technical feasibility
- Operational feasibility

ECONOMICAL FEASIBILITY

This study is carried out to check the economic impact that the system will have on the organization. The amount of funds that the company can pour into the research and development of the system is limited. The expenditures must be justified. Thus the developed system as well within the budget and this was achieved because most of the technologies used are freely available. Only the customized products had to be purchased.

TECHNICAL FEASIBILITY

This study is carried out to check the technical feasibility, that is, the technical requirements of the system. Any system developed must not have a high demand on the available technical resources. This will lead to high demands being placed on the client. The developed system must have modest requirements, as only minimal or null changes are required for implementing this system.

OPERATIONAL FEASIBILITY

The aspect of the study is to check the level of acceptance of the system by the user. This includes the process of training the user to use the system efficiently. The user must not feel threatened by the system, instead must accept it as a necessity. The level of acceptance by the users solely depends on the methods that are employed to educate the user about the system and to make him familiar with it. His level of confidence must be raised so that he is also able to make some constructive criticism, which is welcomed, as he is the final user of the system

3.2 SOFTWARE REQUIREMENTS SPECIFICATION DOCUMENT

Hardware specifications:

- Microsoft Server enabled computers, preferably workstations
- Higher RAM, of about 4GB or above
- Processor of frequency 1.5GHz or above

Software specifications:

- Python 3.6 and higher
- anaconda software

3.3 SYSTEM USE CASE

Most of the previous works on zero-shot learning focused on images and videos, are totally different from other forms of sensor data. Sensor data are generally noisy, and the noise may change the relationship between the features and the desired output it is not clear which features within the sensor data are useful to recognize activities to train a normal model, one generally needs a large amount of sensor data, but in our case, we consider situations where one has only a small amount of sensor data for training. To address the above formulated research questions Q1, Q2, and Q3, we have designed a representation for human activity recognition through correlating high-level activity-labels by embedding semantic similarity. The description of semantic similarities is based on low-level features captured from event occurrences in the sensor data. For research question Q4, to improve the recognition accuracy, we compared the output using various scenarios to reach the best recognition performance. We summarize our core contributions in this work as follows.

- Designing a method to recognize human activity from non-visual sensor readings instead of traditional methods, which depends on images or video observations.
- Combining both the semantic similarity and zero-shot algorithms for robustly recognizing previously unseen human activities.
- Implementing our approach by using different training and testing samples to enhance and validate the good/high recognition accuracy of previously unseen activities.
- Evaluating the system through the use of two well-known public activity recognition datasets.

CHAPTER 4
DESCRIPTION OF PROPOSED SYSTEM

4.1 SELECTED METHODOLOGY OR PROCESS MODEL

Module 1: Data Gathering and Data Preprocessing

1. Creating Hand Landmark Creator

   We will create a landmark creator function to detect the left and right hand from the frame and create a landmark for the hand. The location of hand landmarks is an important source of information to recognize hand gestures, effectively exploited in a number of recent methods that work from depth maps. A hand landmark model processes a cropped bounding box image and returns three dimensional hand key points on hand. This function reaches the exact location of 21 key points with a 3D hand knuckle coordinate which is driven inside the hand regions detected by regression which will produce the prediction of direct coordinates which is a model of the hand landmark. Each hand-knuckle of the landmark coordinate is composed of x, y, and z where x and y are normalized to [0.0, 1.0] by image width and height, while z represents the depth of the landmark. The depth of the landmark that can be found on the wrist being the ancestor. The closer the landmark is to the camera, the value becomes smaller.

2. Data Gathering

   The process of collecting data is crucial for the development of an effective ML model. The quality and quantity of your dataset has a direct effect on the decisionmaking process of the AI model. And these two factors influence the robustness, precision and performance of AI algorithms. As a consequence, data collection and structuring often takes longer than data model training. We will gather data using webcam only for sign language that will contain alphabets (A-Z) and different gestures, i.e. Hello, Thank you, Eat, Goodbye and many more.

3. Creating Labels for Detections

   Image labeling is a type of data labeling that places emphasis on the identification and labeling of specific details in an image. In computer vision, data labeling involves adding tags to raw data like pictures and videos. Each tag stands for an object class associated with the data. Supervised ML models use labels during

learning to identify a specific object class in unclassified data. It helps these models combine meaning with data, which can help form a model. Image annotation is used to create data sets for computer vision models that are divided into training sets, originally used to train the model and test and validation sets used to assess model performance. Data scientists use the dataset to shape and evaluate their model, then the model can automatically label unlabeled and unseen data.
We will create labels for each image to classify the alphabets and gestures. Image labeling is an essential function of computer vision algorithms. The ultimate goal of machine learning algorithms is to obtain automatic labeling, but to train a model will require a large set of pre-labeled image data.

Methods for image labelling

- **Manual image annotations**
  This is the process of manually defining labels for a complete image or drawing regions in an image and adding text descriptions for each region. Manual annotation is typically supported by tools that enable operators to rotate a large number of images, draw regions on an image, and label and store these data in a standard format which may be used for data training.
- **Automated image annotations**
  Automated annotation tools can help manual annotators by trying to find object limits in an image and providing a starting point for annotation. Automated annotation algorithms are not entirely precise, but they can save time for human annotators by providing at least one partial map of the objects in the image.
- **Synthetic image labeling:**
  It is a precise, cost-effective technique that can replace manual annotations. This involves automatically generating images similar to actual data, in accordance with the criteria defined by the operator. There are three common methods of producing synthetic images:
- **Variational Autoencoders (VAE):**
  These are algorithms that start from existing data, create a new data distribution, and link it to the original space using an encoder-decoder method.
- **Generative Adversarial Networks (GAN):**
  These are models between two neural networks. One neural network tries to create fake images, while the other tries to distinguish actual images from fake ones. Over time, the system can produce photorealistic images which are difficult to distinguish from real images.

- **Neural Radiance Fields (NeRF):**
  This model takes a series of images describing a 3D scene and automatically makes additional new perspectives from the same scene. It works by calculating a 5- dimensional radiation function to generate each voxel on the target image.

Data Preprocessing

Algorithms which learn data are just statistical equations operating on database values. Thus, as the saying goes, "if the garbage enters, the garbage leaves". Your data project can only achieve success if the data entering the machines is of high quality. Data from actual scenarios always contain noise and missing values.This occurs due to manual mistakes, unexpected events, technical problems or various other obstacles. Incomplete and noisy data cannot be consumed by algorithms, as they are generally not designed to handle missing values, and noise causes disruption to the actual sample setup. Data preprocessing is a step in transforming raw data so that problems due to incompleteness, inconsistency and/or lack of proper representation of trends are resolved in order to obtain a data set in a comprehensible format. In our model, for data preprocessing we will convert our labels to categorical values using a One Hot Encoding method. As we will be taking raw webcam frames we won't do any other preprocessing method.

Categorical Encoding

Sometimes the data is in a form that cannot be processed by machines. For example, a column with string values, like names, will not mean anything for a model that depends solely on numbers. We therefore have to process the data to assist the model in interpreting them. It is referred to as categorical encoding. There are multiple ways in which we can encode categories. Here we use One Hot Encoding method.

When data is not in an inherent order, we can use only one hot encoding. A hot encoding generates a column for each category and assigns a positive value, 1, in any row within that category, and 0 when not present. The downside is that multiple features are generated from a single feature, which makes the data large. It doesn't matter if we don't have too many features.

Module 2: Model Creation and Training

1. **Alphabet Detection Model**
   The objective is to recognize all 26 Alphabets with the aid of hand gestures through a web camera. We will create an Alphabet detection Model using Tensorflow Dense

model to detect angled and normal alphabets to train our dataset that stands for alphabets. This deep learning model will recognize the alphabet with gestures captured in real time via a webcam. The dense layer is a neural network with a deep connection, which means that each neuron in the dense layer receives the input of all neurons from its previous layer. Dense Layer does a matrix-vector multiplication, and the values used in the matrix are parameters that can be trained and updated with the help of backpropagation. The output produced by the dense layer is a vector of dimension 'n'. Dense Layer is used to change the dimensions, rotate, scale and translate the vector.

2. **Gesture Detection Model**

   We will create a gesture detection Model using Tensorflow Dense model to detect sign language hand gestures to train our dataset that stands for hand gestures. This deep learning model will recognize the sign language with hand gestures captured in real time via a webcam.

   Tensorflow dense is the kind of layer and function available in neural networks whilst implementing artificial intelligence and deep learning in a python programming language. There are deep connections that exist between the neurons in the neural network in dense layers. The pattern they follow is such that each individual neuron gets data input from all the neurons of the previous layer, forming the complex model. While on the other end, dense is also a function used in the neural networks of TensorFlow, which produces the output by applying activation of the dot of Kernel and input and adding the bias effect to it. At the same time, dense is also a function used in TensorFlow neural networks which produces the output by applying kernel point activation and input and adding bias effect.

3. **Data Splitting**

   Data splitting is the time at which data is divided into two or more subsets. Typically, with a two-part split, one part is used to evaluate or test the data and another to form the model. Data splitting is an important part of data science, particularly for the creation of data-based models. This technique ensures that data and process models that utilize data models, such as machine learning, are accurately created.

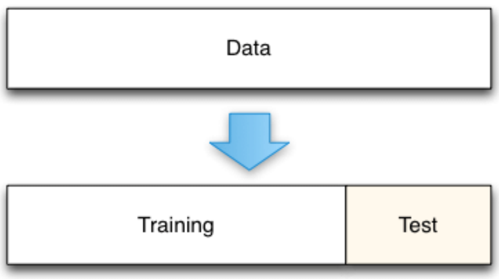


Fig 4.1 Splitting Of Dataset

In a basic two-part split, the training dataset is used for training and model development. Training sets are routinely used to estimate different parameters or to compare the performance of different models. The test data set is used at the end of the training. Training and test data are compared to make sure the final model is working properly. When using machine learning, data is usually divided into three or more sets. In three sets, the extra set is the development set, which is used to modify the parameters of the learning process. Organizations and data modelers may elect to separate split data using data sampling methods, such as the following three methods:

- Random sampling: This data sampling method protects the data modeling process against bias to different data characteristics that may occur. However, random splitting can pose problems in terms of uneven distribution of data.
- Stratified random sampling: It selects random data samples from specific parameters. Ensures data is properly distributed across training and test sets.
- Nonrandom sampling: This approach is generally used when data modelers are looking for the latest data as a test set.

With data splitting, organizations do not have to choose between using data for statistical analysis and analytics, as the same data can be used in the various processes

Training Model Model training is the lifecycle phase of data science development during which practitioners attempt to figure out the best combination of weight and bias to a machine learning algorithm to minimize a loss function on the prediction range. The goal of model development is to construct the best mathematical representation of the relationship between data features and a target label (in supervised learning) or among the features themselves (unsupervised learning). Model training is the key step of machine learning leading to a model ready for validation, testing and deployment. The performance of the model determines the quality of the applications which are constructed using it. The quality of the training data and the training algorithm are significant assets in the training phase of the model. Training information is usually split for training, validation and testing. The training algorithm is selected as a function of the end-use case. There are a number of trade-off points for choosing the best algorithm – model complexity, interpretability, performance, computational requirements, etc. All these aspects of model training make it a process that is both involved and important for the overall development cycle of machine learning. We will load our CUDA GPU to train our model. CUDA is a parallel computing platform and programming model developed by NVIDIA for general computing in its own GPUs (graphics processing units). CUDA allows developers to accelerate compute-intensive applications by leveraging the power of GPUs for the parallelised part of the computation. Although there have been other APIs proposed for GPUs, such as OpenCL, and there are competitive GPUs from other companies, such as AMD, the combination of CUDA and NVIDIA GPUs dominates a number of application domains, including deep learning, and is a foundation for some of the fastest computers in the world.

Saving the Best Model Saving our machine learning models is an important step in the machine learning workflow, so we can reuse them in the future. For example, it is very likely that we will have to compare models to determine which champion model to use in production.

Saving the models while they are being trained facilitates this process. The alternative would be to form the model whenever it is to be used, which can greatly affect productivity, especially if the model takes a long time to form.

Module 3: Creating Node JS Web App

An important part of building a machine learning model is to share the model we have built with others. No matter how many models we create, if they remain offline, very few people will be able to see what we are achieving. That's why we should deploy our models, so that anyone can play with them through a nice User Interface (UI). For this system, we will create a Node JS webapp to take webcam input using openCV and detect alphabets and gestures using our trained model. Nodejs is a server-side platform powered by Google Chrome JavaScript Engine (V8 Engine). Node.js is a platform built on Chrome JavaScript runtime to build fast, scalable network applications with ease. Node.js uses a non-blocking, event-driven I/O model that makes it lightweight and efficient, perfect for real-time, data-intensive applications that operate on distributed devices. It is an open source, cross-platform runtime environment for developing server-side and networking applications. Node.js applications are written in JavaScript and may run in the Node.js runtime environment on OS X, Microsoft Windows and Linux. Node.js also provides a rich library of different JavaScript modules that simplifies the development of web applications using Node.js to a great degree. Node.js = Runtime Environment + JavaScript Library These are just a few of the key features that make Node.js the first choice for software architects:

- Asynchronous and Event Driven: All APIs of the Node.js library are asynchronous, meaning non-blocking. This basically means that a Node.js-based server never waits until an API returns data. The server moves to the next API after calling it and a Node.js event notification mechanism assists the server to get a response from the previous API call.
- Very Fast: Being constructed on Google Chrome V8 JavaScript Engine, Node.js library is very fast in code execution.
- Single Threaded but Highly Scalable: Node.js has a single threaded model with an event loop. The event mechanism helps the server to respond in a nonblocking manner and makes the server highly scalable as opposed to traditional servers that create limited threads to manage requests. Node.js uses a single threaded program and the same program can serve a lot more requests than traditional servers like Apache HTTP Server.
- No Buffering: No Node.js application buffers the data. These applications simply generate data by chunks.

License: Node.js is licensed by MIT

Proposed System

- The proposed machine has a "Gesture Speech Module".
- Gesture to Speech module takes gestures as input in the form of image/video and produces speech with the corresponding individual as output.
- The proposed research pursuits to increase a machine that acknowledges static sign gestures and interprets them into appropriate words.
- The cause of the gadget is to function as a learning tool for those who want to research more about the fact of sign language, which include alphabets, numbers and common place country symbols.
- In the proposed gadget, the picture dataset includes a static gesture language.
- The information set is then pre-processed, and then it will likely be used to shape a wiped clean photo.
- The wiped clean photographs will then be tested, and any necessary actions will need to be taken, consisting of identifying the photograph, changing all resolutions to the identical, and so on.
- The version will then be built the use of OpenCV (Computer Vision) where all the layers used and different factors can be defined to assist us construct the version.

- Finally, the quality version may be kept in making the prediction.
- A prediction version predicts output traits based totally on input statistics.

Advantages of the Proposed System

- Live video inputs are detected
- Used gesture datasets
- Excessive accuracy 90%
- An green gadget with Open CV

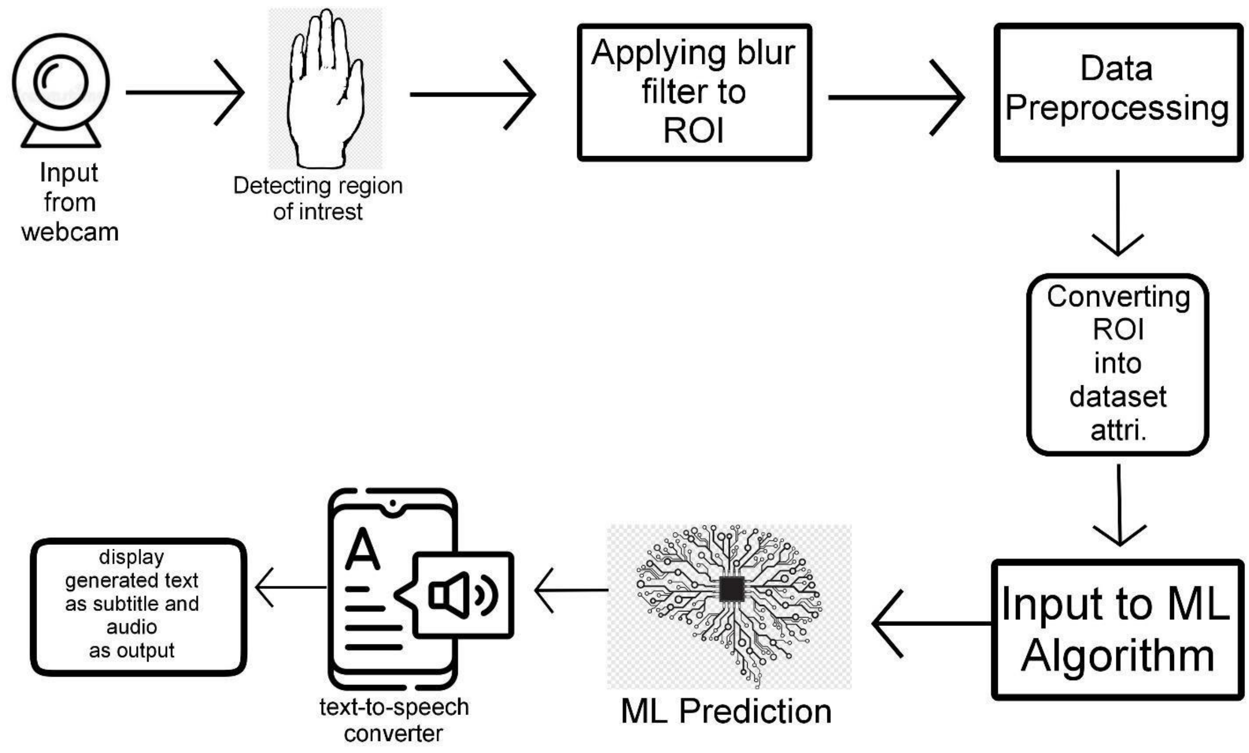

Input from webcam
Detecting region of intrest
Applying blur filter to ROI
Data Preprocessing
Converting ROI into dataset attri.
Input to ML Algorithm
ML Prediction
text-to-speech converter
display generated text as subtitle and audio as output

Fig.4.2 Architecture of Proposed System

4.2 DESCRIPTION OF SOFTWARE FOR IMPLEMENTATION AND TESTING PLAN OF THE PROPOSED SYSTEM

Introduction to Keras VGG16

Keras VGG16 is a deep learning model which was available with pre-trained weights. The Keras VGG16 model is used in feature extraction, fine-tuning, and prediction models. By using Keras VGG16 weights are downloaded automatically by instantiating the model of Keras and this model is stored in Keras/model directory. As per the instantiation, the

Keras model will build according to the image data format which set up our Keras VGG16 configuration files.

Key Takeaways

- Transfer learning is referring the process where the model of Keras VGG16 is trained by using specified problems.
- In deep transfer learning, the model of a neural network was first trained by using a similar problem that we are solving in that specified neural model.

What is Keras VGG16?

The Keras VGG16 is nothing but the architecture of the convolution neural net which was used in ILSVR. The Keras VGG16 model is considered the architecture of the vision model. The very important thing regarding VGG16 is that instead of a large parameter it will focus on the convolution layers. It is following the arrangement of max pool layers and convolution which was consistent throughout the architecture. It is referring the 16 layers which contain weights. The Keras VGG16 network is very large, it will contain millions of parameters. It is implemented on a dataset of python.

4.3 PROJECT MANAGEMENT PLAN

- **Convolutional layer:** This layer will be computing the output of nodes which was connected to input matrix regions.
- **Activation layer:** This layer will be determining whether the input node is fired by using input data. The volume of dimensions is unchanged by using this layer.
- **Pooling layer:** While using this layer down sampling strategy is applied while reducing the weight and height of the volume.
- **Fully connected layer:** The volume of output is passed by using the nodes of the fully connected layer. The probability class is computed and outputted into the 3D array by using dimensions.

1. Keras CNN predicting the food labels

In the below example, we are loading the model for generating the predictions and calculating accuracy which was used for comparing the performance as follows. While using it we need to install the keras in our system. In the below example, we are installing the same by using the pip command as follows.

2. How we can transfer learning work?

Transfer learning will be resolving the limitation of the learning paradigm. Transfer learning will be giving the ability for sharing features across tasks.

3. Domain and task

Domain and task are defined in a domain and task. In the domain, we are defining the image classification of our task to classify the images. If suppose we are using the CNN, then it will already be optimized and also it will be trained for task and domain. We can utilize the model which was pertained.

4. How we can utilize the model of VGG16?

VGG16 is a CN network that was trained into the collection dataset. The image net dataset will contain images of different types of vehicles. We need to import the model which was pre-trained onto the dataset of image net.There are two types of approaches to keras VGG16:

4.4 FINANCIAL REPORT ON ESTIMATED COSTING

- **Feature extraction approach:** We can use the pre trained model architecture for creating the new dataset from the input images. By using this type of approach, we are importing the pooling and convolutional layers. We are passing the images through VGG16.
- **Fine tuning approach:** This type of approach is used to employ the fine-tuning strategy. The goal of this strategy is to allow the layers of pretrained. In the approach of feature

extraction, we have used pretrained layers. We are passing our image dataset from weights outputting.

6. Transfer learning for the classification of food

The VGG16 model is easily downloaded by using the keras API. We need to import the function of pre-processing with the VGG16 model. The keras VGG16 model is trained by using pixels value which was ranging from 0 to 255. Other models contain different normalization schemes into it.

7. Training and testing data preparation

In the below example, we are first importing the libraries.

7. Pre-trained layers for fine-tuning

In the below example, we are using the pre trained layer for the feature extraction.

9. Comparing models

In the below example, we are comparing the model which was generated in the above case

Keras VGG16 Architecture

In the VGG16 architecture, there are 13 layers available, five are the max pooling, and three are dense layers. Below figure shows keras VGG16 architecture.The training data is forwarded through the network and the same is stopped in the FC layers. In some of the network of the case is obtaining the accuracy. To use it we need to install the tensor flow in our system.

4.5 TRANSITION/SOFTWARE TO OPERATION PLAN

- **User**
- **HAR System**
- **VGG16**
- **Transfer Learning**

Modules Description

User

The User can start the project by running mainrun.py file. User has to give –input (Video file path).The open cv class Video Capture means primary camera of the system, Video Capture means secondary camera of the system. Video Capture (Video file path) means without camera we can load the video file from the disk. Vgg16, Vgg19 has programmatically configured. User can change the model selection in the code and can run in multiple ways.

HAR System

Video-based human activity recognition can be categorized as vision-based according. The vision based method make use of RGB or depth image. It does not require the user to carry any devices or to attach any sensors on the human. Therefore, this methodology is getting more consideration nowadays, consequently making the HAR framework simple and easy to be deployed in many applications. We first extracted the frames for each activities from the videos. Specifically, we use transfer learning to get deep image features and trained machine learning classifiers.

HAR datasets are a vivid variety of qualities based upon their parameters, such as RGB, RGB-D(Depth), Multi view, recorded in a controlled environment. Other parameters are – recorded “In the wild,” annotated with a complete sentence, annotated with only action label datasets, etc, such as the source of data collection, number of actions, video clips, nature of datasets, and released year to show the progress in this area. We observe that most of the HAR datasets could not become a popular choice among computer-vision researchers due to their over simplicity, small size, and unsatisfactory performance. However, there is no such thing as the most accurate standard datasets, i.e., on which researchers measure the HAR method to set as a benchmark, but of course, as we observe UCF101 and are the dominating datasets for researcher’s interest. Also, the actions played in the recorded clips are, by various individuals, while in other datasets, the activities and actions are usually performed by one actor only.

VGG16

VGG16 is a convolutional neural network model. Deep Convolutional Networks for Large-Scale Image Recognition”. The model achieves 92.7% top-5 test accuracy in ImageNet, which is a dataset of over 14 million images belonging to 1000 classes. It was one of the famous model submitted to ILSVRC-2014. It makes the improvement over AlexNet by replacing large kernel-sized filters (11 and 5 in the first and second convolutional layer, respectively) with multiple 3×3 kernel-sized filters one after another. VGG16 was trained for weeks and was using NVIDIA Titan Black GPU's.

VGG-16 Architecture

The input to the network is an image of dimensions *(224, 224, 3)*. The first two layers have *64* channels of a *3*3* filter size and the same padding. Then after a max pool layer of stride *(2, 2)*, two layers have convolution layers of 128 filter size and filter size *(3, 3)*. This is followed by a max-pooling layer of stride *(2, 2)* which is the same as the previous layer. Then there are *2* convolution layers of filter size *(3, 3)* and *256* filters. After that, there are *2* sets of *3* convolution layers and a max pool layer. Each has *512* filters of *(3, 3)* size with the same padding. This image is then passed to the stack of two convolution layers. In these convolution and max-pooling layers, the filters we use are of the size *3*3* instead of *11*11* in AlexNet and *7*7* in ZF-Net. In some of the layers, it also uses *1*1* pixel which is used to manipulate the number of input channels.
There is a padding of *1-pixel* (same padding) done after each convolution layer to prevent the spatial feature of the image.

After the stack of convolution and max-pooling layer, we got a *(7, 7, 512)* feature map. We flatten this output to make it a *(1, 25088)* feature vector. After this there is *3 fully* connected layer, the first layer takes input from the last feature vector and outputs a *(1, 4096)* vector, the second layer also outputs a vector of size *(1, 4096)* but the third layer output a *1000* channels for *1000* classes of ILSVRC challenge i.e. 3rd fully connected layer is used to implement softmax function to classify 1000 classes. All the hidden layers use ReLU as its activation function. ReLU is more computationally efficient because it results in faster learning and it also decreases the likelihood of vanishing gradient problems.

Transfer Learning

Transfer learning is a machine learning method where a model developed for a task is reused as the starting point for a model on a second task. It is a popular approach in deep learning where pre-trained models are used as the starting point on computer vision and natural language processing tasks given the vast compute and time resources required to develop neural network models on these problems and from the huge jumps in skill that they provide on related problems. In this post, you will discover how you can use transfer learning to speed up training and improve the performance of your deep learning model.

Transfer learning is a method of transferring *knowledge* that a model has learned from earlier extensive training to the current model. The deep network models can be trained with significantly less data with transfer learning. It has been used to reduce training time and improve accuracy of the model. In this work, we use *transfer learning* to leverage the knowledge gained from large-scale dataset such as ImageNet. We first extract the frames for each activities from the videos. We use *transfer learning* to get deep image features and trained machine learning classifiers. For all CNN models, pre-trained weights on ImageNet are used as starting point for transfer learning. ImageNet [6] is a dataset containing 20000 categories of activities. The knowledge is transferred from pre-trained weights on ImageNet to Weizmann dataset, since set of activities recognised in this work fall within the domain of ImageNet. The features are extracted from the penultimate layer of CNNs.

The basic idea of transfer learning is as shown in the figure given below

CHAPTER 5
IMPLEMENTATION DETAILS

5.1 DEVELOPMENT AND DEPLOYMENT SETUP

For each record it is provided

- - Triaxial acceleration from the accelerometer (total acceleration) and the estimated body acceleration.
- - Triaxial Angular velocity from the gyroscope.
- - A 561-feature vector with time and frequency domain variables.
- - Its activity label.
- - An identifier of the subject who carried out the experiment.

The dataset includes the following files

- - 'README.txt'
- - 'features_info.txt': Shows information about the variables used on the feature vector.
- - 'features.txt': List of all features.
- - 'activity_labels.txt': Links the class labels with their activity name.
- - 'train/X_train.txt': Training set.
- - 'train/y_train.txt': Training labels.
- - 'test/X_test.txt': Test set.
- - 'test/y_test.txt': Test labels.
- The following files are available for the train and test data. Their descriptions are equivalent.
- - 'train/subject_train.txt': Each row identifies the subject who performed the activity for each window sample. Its range is from 1 to 30.
- - 'train/Inertial Signals/total_acc_x_train.txt': The acceleration signal from the smartphone accelerometer X axis in standard gravity units 'g'. Every row shows a 128-element vector.

The same description applies for the 'total_acc_x_train.txt' and 'total_acc_z_train.txt' files for the Y and Z axis.

- - 'train/Inertial Signals/body_acc_x_train.txt': The body acceleration signal obtained by subtracting the gravity from the total acceleration.
- - 'train/Inertial Signals/body_gyro_x_train.txt': The angular velocity vector measured by the gyroscope for each window sample. The units are radians/second.

CALCULATIONS USED FOR DATA SETS

- - Features are normalized and bounded within [-1,1].
- - Each feature vector is a row on the text file.
- - The units used for the accelerations (total and body) are 'g's (gravity of earth -> 9.80665 m/seg2).
- - The gyroscope units are rad/seg.
- - A video of the experiment including an example of the 6 recorded activities with one of the participants can be seen in the following link: http://www.youtube.com/watch?v=XOEN9W05_4A

Installations Done on System

- Python-3.8.5-64-bit
- Python Launcher
- Notepad++ 8.4.6
- Notepad++ GPS Signature-8.4.6
- Notepad++ Installer-8.4.6
- Open CV Binaries in Python-4.5.5-cp39-cp39-win and 64.whl

Installed Packages

absl-py-1.3.0, astunparse-1.6.3, cachetools-4.2.4, certifi-2022.12.7, charset-normalizer 2.1.1,contourpy-1.0.6,cycler-0.11.0,fonttools-4.38.0,gast-0.3.3,google-auth1.35.0,google-auth-oauthlib-0.4.6,google-pasta-0.2.0,grpcio-1.51.1,h5py-2.10.0,idna-3.4,importlib-metadata-6.0.0,imutils-0.5.4,keras-2.11.0,KerasPreprocessing1.1.2,pandas1.5.2, Tensorboard-2.2.2, tensorboard-plugin-wit-1.8.1

5.2 ALOGORITHMS

Decision Tree Algorithm:

- Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.
- In a Decision tree, there are two nodes, which are the Decision Node and Leaf Node. Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches.
- The decisions or the test are performed on the basis of features of the given dataset.
- It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions.
- It is called a decision tree because, similar to a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure.
- In order to build a tree, we use the CART algorithm, which stands for Classification and Regression Tree algorithm.
- A decision tree simply asks a question, and based on the answer (Yes/No), it further split the tree into subtrees.

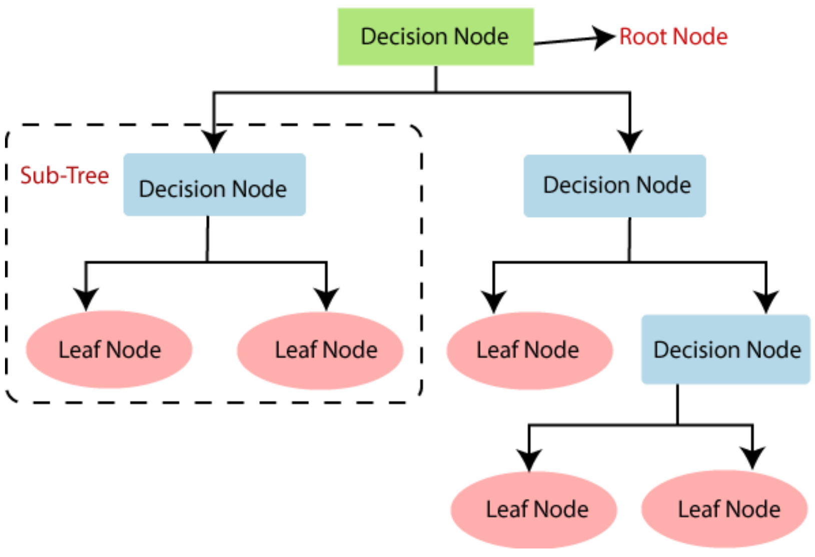


Fig 5.1 Decision Tree Algorithm flow

Decision Tree Terminologies

- Root Node: Root node is from where the decision tree starts. It represents the entire dataset, which further gets divided into two or more homogeneous sets.
- Leaf Node: Leaf nodes are the final output node, and the tree cannot be segregated further after getting a leaf node.
- Splitting: Splitting is the process of dividing the decision node/root node into sub-nodes according to the given conditions.
- Branch/Sub Tree: A tree formed by splitting the tree.
- Pruning: Pruning is the process of removing the unwanted branches from the tree.
- Parent/Child node: The root node of the tree is called the parent node, and other nodes are called the child nodes.

In a decision tree, for predicting the class of the given dataset, the algorithm starts from the root node of the tree. This algorithm compares the values of root attribute with the record (real dataset) attribute and, based on the comparison, follows the branch and jumps to the next node. For the next node, the algorithm again compares the attribute value with the other sub-nodes and move further. It continues the process until it reaches the leaf node of the tree. The complete process can be better understood using the below algorithm:

- Step-1: Begin the tree with the root node, says S, which contains the complete dataset.
- Step-2: Find the best attribute in the dataset using Attribute Selection Measure (ASM).
- Step-3: Divide the S into subsets that contains possible values for the best attributes.
- Step-4: Generate the decision tree node, which contains the best attribute.
- Step-5: Recursively make new decision trees using the subsets of the dataset created in step -3. Continue this process until a stage is reached where you cannot further classify the nodes and called the final node as a leaf node.

Attribute Selection Measures

While implementing a Decision tree, the main issue arises that how to select the best attribute for the root node and for sub-nodes. So, to solve such problems there is a technique which is called as Attribute selection measure or ASM. By this measurement, we can easily select the best attribute for the nodes of the tree. There are two popular techniques for ASM, which are:

- Information Gain
- Gini Index

1. Information Gain:

- Information gain is the measurement of changes in entropy after the segmentation of a dataset based on an attribute.
- It calculates how much information a feature provides us about a class.
- According to the value of information gain, we split the node and build the decision tree.
- A decision tree algorithm always tries to maximize the value of information gain, and a node/attribute having the highest information gain is split first. It can be calculated using the below formula:

Information Gain= Entropy(S)- [(Weighted Avg) *Entropy(each feature)

Entropy: Entropy is a metric to measure the impurity in a given attribute. It specifies randomness in data. Entropy can be calculated as:

Entropy(s)= -P(yes)log2 P(yes)- P(no) log2 P(no)

Where,

- S= Total number of samples
- P(yes)= probability of yes
- P(no)= probability of no

2. Gini Index:

- Gini index is a measure of impurity or purity used while creating a decision tree in the CART(Classification and Regression Tree) algorithm.
- An attribute with the low Gini index should be preferred as compared to the high Gini index.
- It only creates binary splits, and the CART algorithm uses the Gini index to create binary splits.
- Gini index can be calculated using the below formula:

$$\textbf{Gini Index= 1- } \sum_j P_j^2$$

Pruning: Getting an Optimal Decision tree

Pruning is a process of deleting the unnecessary nodes from a tree in order to get the optimal decision tree.

A too-large tree increases the risk of overfitting, and a small tree may not capture all the important features of the dataset. Therefore, a technique that decreases the size of the learning tree without reducing accuracy is known as Pruning. There are mainly two types of tree pruning technology used:

- Cost Complexity Pruning
- Reduced Error Pruning.

Advantages of the Decision Tree

- It is simple to understand as it follows the same process which a human follow while making any decision in real-life.
- It can be very useful for solving decision-related problems.
- It helps to think about all the possible outcomes for a problem.
- There is less requirement of data cleaning compared to other algorithms.

Disadvantages of the Decision Tree

- The decision tree contains lots of layers, which makes it complex.
- It may have an overfitting issue, which can be resolved using the Random Forest algorithm.
- For more class labels, the computational complexity of the decision tree may increase.

Python Implementation of Decision Tree

- Data Pre-processing step
- Fitting a Decision-Tree algorithm to the Training set
- Predicting the test result
- Test accuracy of the result(Creation of Confusion matrix)
- Visualizing the test set result.

VGG-16(also called Oxford Net)

VGG16 proved to be a significant milestone in the quest of mankind to make computers "see" the world. A lot of effort has been put into improving this ability under the discipline of Computer Vision (CV) for a number of decades. VGG16 is one of the significant innovations that paved the way for several innovations that followed in this field. The input to any of the network configurations is considered to be a fixed size 224 x 224 image with three channels – R, G, and B. The only pre-processing done is normalizing the RGB values for every pixel. This is achieved by subtracting the mean value from every pixel.

Image is passed through the first stack of 2 convolution layers of the very small receptive size of 3 x 3, followed by ReLU activations. Each of these two layers contains 64 filters. The convolution stride is fixed at 1 pixel, and the padding is 1 pixel. This configuration preserves the spatial resolution, and the size of the output activation map is the same as the input image dimensions. The activation maps are then passed through spatial max pooling over a 2 x 2-pixel window, with a stride of 2 pixels. This halves the size of the activations. Thus the size of the activations at the end of the first stack is 112 x 112 x 64.

The activations then flow through a similar second stack, but with 128 filters as against 64 in the first one. Consequently, the size after the second stack becomes 56 x 56 x 128. This is followed by the third stack with three convolutional layers and a max pool layer. The no. of filters applied here are 256, making the output size of the stack 28 x 28 x 256. This is followed by two stacks of three convolutional layers, with each containing 512 filters. The output at the end of both these stacks will be 7 x 7 x 512.

The stacks of convolutional layers are followed by three fully connected layers with a flattening layer in-between. The first two have 4,096 neurons each, and the last fully connected layer serves as the output layer and has 1,000 neurons corresponding to the 1,000 possible classes for the ImageNet dataset. The output layer is followed by the Softmax activation layer used for categorical classification.

CHAPTER 6

RESULTS AND DISCUSSION

RESULTS

The main objective of this project is to detect hand gestures and provide corresponding words as text and audio as output which has been successfully achieved by this project model.

DISCUSSION

This module can be included in online communication live streaming platforms so that it bridges the communication gap between sensory impaired people and normal people. Training this model with much more bigger datasets can eventually increases the accuracy.

CHAPTER 7

CONCULSION

7.1 CONCLUSION

This project introduces a Computer vision approach for the recognition and classification of sign language using deep learning Each system was trained using 50 × 50 images of each word gesture. Unlike the other approaches, this approach yields better accuracy and considerably low false positives. This project was implemented in real time using web camera and this project yields better accuracy than the existing system.

7.2 FUTURE WORK

The proposed sign language recognition system used to recognize gestures can be further extended to recognize facial expressions.

1. Instead of displaying letter labels, it will be more appropriate to display sentences as a more appropriate translation of language.
2. This also increases readability.
3. The scope of different sign languages can be increased. More training data can be added to detect the letter with more accuracy.
4. This project can further be extended to convert the signs to speech.
5. Captions can be added to the gestures recognized.

REFERENCE

[1] Soma Shrenika, Myneni Madhu Bala, “Sign Language Recognition Using Template Matching Technique”, International Conference on Computer Science, Engineering and Applications (ICCSEA), 2020.

[2] H Muthu Mariappan, V Gomathi, “Real-Time Recognition of Indian Sign Language”, International Conference on Computational Intelligence in Data Science (ICCIDS), 2019.

[3] Wanbo Li, Hang Pu, Ruijuan Wang, “Sign Language Recognition Based on Computer Vision”, IEEE International Conference on Artificial Intelligence and Computer Applications (ICAICA), 2021.

[4] Pushkar Kurhekar, Janvi Phadtare, Sourav Sinha, Kavita P. Shirsat, “Real Time Sign Language Estimation System”, 3rd International Conference on Trends in Electronics and Informatics (ICOEI), 2019.

[5] Ruchi Gajjar, Jinalee Jayeshkumar, Shubham Kumar, Aniket Saha, Suman Yadav, “Indian Sign Language Gesture Recognition in Real-Time using Convolutional Neural Networks”, 8th International Conference on Signal Processing and Integrated Networks (SPIN), 2021.

[6] Sai Nikhilesh Reddy Karna, Jai Surya Kode, Suneel Nadipalli, Sudha Yadav, “American Sign Language Static Gesture Recognition using Deep Learning and Computer Vision”, 2nd International Conference on Smart Electronics and Communication (ICOSEC), 2021.

[7] Shirin Sultana Shanta, Saif Taifur Anwar, Md. Rayhanul Kabir, “Bangla Sign Language Detection Using SIFT and CNN”, 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT), 2018.

[8] Rajiv Ranjan, B Shivalal Patro, Mohammad Daim Khan, Manas Chandan Behera, Raushan Kumar, Utsav Raj, “A Review on Sign Language Recognition Systems”, IEEE 2nd International Conference on Applied Electromagnetics, Signal Processing, & Communication (AESPC), 2022.

[9] Kohsheen Tiku, Jayshree Maloo, Aishwarya Ramesh, Indra R, “Real-time Conversion of Sign Language to Text and Speech”, Second International Conference on Inventive Research in Computing Applications (ICIRCA), 2020.

[10] Vishwa Hariharan Iyer, U.M Prakash, Aashrut Vijay, P. Sathishkumar, “Sign Language Detection using Action Recognition”, 2nd International Conference on Advance Computing and Innovative Technologies in Engineering (ICACIT).

[11] Vasileios. Apostolidis-Afentoulis, SVM Classification with Linear and RBF kernels, 2015.

[12] Kumar Pradeep, Gauba Himaanshu, Roy Partha and Dogra Debi, "A Multimodal Framework for Sensor based Sign Language Recognition", Neurocomputing, vol. 259, 2017.

[13] Dumitrescu and Costin-Anton. Boiangiu, "A Study of Image Upsampling and Downsampling Filters", Computers, vol. 8, no. 30, 2019.

[14] Singh Sanjay, Pai Suraj, Mehta Nayan, Varambally Deepthi, Kohli Pritika and T. Padmashri, Computer Vision Based Sign Language Recognition System, 2019.

[15] Banjoko Alabi, Yahya Waheed, Garba Mohammed, Olaniran Oyebayo, Dauda Kazeem and Olorede Kabir, SVM Paper in Tibiscus Journal 2016.

APPENDIX

A) SOURCE CODE

Cnnlstm.py

```
# convlstm model
from numpy import mean
from numpy import std
from numpy import dstack
from pandas import read_csv
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers import Dropout
from keras.layers import LSTM
from keras.layers import TimeDistributed
from keras.layers import ConvLSTM2D
from keras.utils import to_categorical
from matplotlib import pyplot

# load a single file as a numpy array
def load_file(filepath):
    dataframe = read_csv(filepath, header=None, delim_whitespace=True)
    return dataframe.values

# load a list of files and return as a 3d numpy array
def load_group(filenames, prefix=''):
    loaded = list()
    for name in filenames:
            data = load_file(prefix + name)
            loaded.append(data)
    # stack group so that features are the 3rd dimension
    loaded = dstack(loaded)
    return loaded

# load a dataset group, such as train or test
def load_dataset_group(group, prefix=''):
    filepath = prefix + group + '/Inertial Signals/'
    # load all 9 files as a single array
    filenames = list()
    # total acceleration
    filenames    +=    ['total_acc_x_'+group+'.txt',    'total_acc_y_'+group+'.txt',
'total_acc_z_'+group+'.txt']
    # body acceleration
    filenames    +=    ['body_acc_x_'+group+'.txt',    'body_acc_y_'+group+'.txt',
'body_acc_z_'+group+'.txt']
    # body gyroscope
    filenames += ['body_gyro_x_'+group+'.txt', 'body_gyro_y_'+group+'.txt',
'body_gyro_z_'+group+'.txt']
    # load input data
    X = load_group(filenames, filepath)
    # load class output
    y = load_file(prefix + group + '/y_'+group+'.txt')
    return X, y

# load the dataset, returns train and test X and y elements
def load_dataset(prefix=''):
    # load all train
    trainX, trainy = load_dataset_group('train', prefix + 'HARDataset/')
    print(trainX.shape, trainy.shape)
    # load all test
    testX, testy = load_dataset_group('test', prefix + 'HARDataset/')
    print(testX.shape, testy.shape)
    # zero-offset class values
    trainy = trainy - 1
    testy = testy - 1
    # one hot encode y
    trainy = to_categorical(trainy)
    testy = to_categorical(testy)
    print(trainX.shape, trainy.shape, testX.shape, testy.shape)
    return trainX, trainy, testX, testy

# fit and evaluate a model
def evaluate_model(trainX, trainy, testX, testy):
    # define model
    verbose, epochs, batch_size = 0, 25, 64
    n_timesteps, n_features, n_outputs = trainX.shape[1], trainX.shape[2], trainy.shape[1]
    # reshape into subsequences (samples, time steps, rows, cols, channels)
    n_steps, n_length = 4, 32
    trainX = trainX.reshape((trainX.shape[0], n_steps, 1, n_length, n_features))
    testX = testX.reshape((testX.shape[0], n_steps, 1, n_length, n_features))
    # define model
    model = Sequential()
    model.add(ConvLSTM2D(filters=64, kernel_size=(1,3), activation='relu',
input_shape=(n_steps, 1, n_length, n_features)))
    model.add(Dropout(0.5))
    model.add(Flatten())
    model.add(Dense(100, activation='relu'))
    model.add(Dense(n_outputs, activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
    # fit network
    model.fit(trainX, trainy, epochs=epochs, batch_size=batch_size, verbose=verbose)
    # evaluate model
    _, accuracy = model.evaluate(testX, testy, batch_size=batch_size, verbose=0)
    return accuracy

# summarize scores
```

```
def summarize_results(scores):
    print(scores)
    m, s = mean(scores), std(scores)
    print('Accuracy: %.3f%% (+/-%.3f)' % (m, s))

# run an experiment
def run_experiment(repeats=10):
    # load data
    trainX, trainy, testX, testy = load_dataset()
    # repeat experiment
    scores = list()
    for r in range(repeats):
        score = evaluate_model(trainX, trainy, testX, testy)
        score = score * 100.0
        print('>#%d: %.3f' % (r+1, score))
        scores.append(score)
    # summarize results
    summarize_results(scores)

# run the experiment
run_experiment()
```

cnn_lstm_model.py

```
# cnn lstm model
from numpy import mean
from numpy import std
from numpy import dstack
from pandas import read_csv
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers import Dropout
from keras.layers import LSTM
from keras.layers import TimeDistributed
from keras.layers.convolutional import Conv1D
from keras.layers.convolutional import MaxPooling1D
from keras.utils import to_categorical
from matplotlib import pyplot

# load a single file as a numpy array
def load_file(filepath):
    dataframe = read_csv(filepath, header=None, delim_whitespace=True)
    return dataframe.values

# load a list of files and return as a 3d numpy array
def load_group(filenames, prefix=''):
    loaded = list()
    for name in filenames:
        data = load_file(prefix + name)
        loaded.append(data)
    # stack group so that features are the 3rd dimension
    loaded = dstack(loaded)
    return loaded

 # load a dataset group, such as train or test
def load_dataset_group(group, prefix=''):
    filepath = prefix + group + '/Inertial Signals/'
    # load all 9 files as a single array
    filenames = list()
    # total acceleration
    filenames += ['total_acc_x_'+group+'.txt', 'total_acc_y_'+group+'.txt',
'total_acc_z_'+group+'.txt']
    # body acceleration
    filenames += ['body_acc_x_'+group+'.txt', 'body_acc_y_'+group+'.txt',
'body_acc_z_'+group+'.txt']
    # body gyroscope
    filenames += ['body_gyro_x_'+group+'.txt', 'body_gyro_y_'+group+'.txt',
'body_gyro_z_'+group+'.txt']
    # load input data
    X = load_group(filenames, filepath)
    # load class output
    y = load_file(prefix + group + '/y_'+group+'.txt')
    return X, y

# load the dataset, returns train and test X and y elements
def load_dataset(prefix=''):
    # load all train
    trainX, trainy = load_dataset_group('train', prefix + 'HARDataset/')
    print(trainX.shape, trainy.shape)
    # load all test
    testX, testy = load_dataset_group('test', prefix + 'HARDataset/')
    print(testX.shape, testy.shape)
    # zero-offset class values
    trainy = trainy - 1
    testy = testy - 1
    # one hot encode y
    trainy = to_categorical(trainy)
    testy = to_categorical(testy)
    print(trainX.shape, trainy.shape, testX.shape, testy.shape)
    return trainX, trainy, testX, testy

# fit and evaluate a model
def evaluate_model(trainX, trainy, testX, testy):
    # define model
    verbose, epochs, batch_size = 0, 25, 64
    n_timesteps, n_features, n_outputs = trainX.shape[1], trainX.shape[2], trainy.shape[1]
    # reshape data into time steps of sub-sequences
    n_steps, n_length = 4, 32
    trainX = trainX.reshape((trainX.shape[0], n_steps, n_length, n_features))
    testX = testX.reshape((testX.shape[0], n_steps, n_length, n_features))
    # define model
    model = Sequential()
    model.add(TimeDistributed(Conv1D(filters=64, kernel_size=3, activation='relu'),
input_shape=(None,n_length,n_features)))
    model.add(TimeDistributed(Conv1D(filters=64, kernel_size=3, activation='relu')))
    model.add(TimeDistributed(Dropout(0.5)))
    model.add(TimeDistributed(MaxPooling1D(pool_size=2)))
    model.add(TimeDistributed(Flatten()))
    model.add(LSTM(100))
    model.add(Dropout(0.5))
    model.add(Dense(100, activation='relu'))
    model.add(Dense(n_outputs, activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
    # fit network
    model.fit(trainX, trainy, epochs=epochs, batch_size=batch_size, verbose=verbose)
    # evaluate model
    _, accuracy = model.evaluate(testX, testy, batch_size=batch_size, verbose=0)
    return accuracy

# summarize scores
def summarize_results(scores):
    print(scores)
    m, s = mean(scores), std(scores)
    print('Accuracy: %.3f%% (+/-%.3f)' % (m, s))

# run an experiment
def run_experiment(repeats=10):
    # load data
    trainX, trainy, testX, testy = load_dataset()
    # repeat experiment
    scores = list()
    for r in range(repeats):
            score = evaluate_model(trainX, trainy, testX, testy)
            score = score * 100.0
            print('>#%d: %.3f' % (r+1, score))
            scores.append(score)
    # summarize results
    summarize_results(scores)

# run the experiment
run_experiment()
```

mylstm.py

```
# lstm model
from numpy import mean
from numpy import std
from numpy import dstack
from pandas import read_csv
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers import Dropout
from keras.layers import LSTM
from keras.utils import to_categorical
from matplotlib import pyplot

# load a single file as a numpy array
def load_file(filepath):
    dataframe = read_csv(filepath, header=None, delim_whitespace=True)
    return dataframe.values

# load a list of files and return as a 3d numpy array
def load_group(filenames, prefix=''):
    loaded = list()
    for name in filenames:
            data = load_file(prefix + name)
            loaded.append(data)
    # stack group so that features are the 3rd dimension
    loaded = dstack(loaded)
    return loaded

# load a dataset group, such as train or test
def load_dataset_group(group, prefix=''):
    filepath = prefix + group + '/Inertial Signals/'
    # load all 9 files as a single array
    filenames = list()
    # total acceleration
    filenames += ['total_acc_x_'+group+'.txt', 'total_acc_y_'+group+'.txt',
'total_acc_z_'+group+'.txt']
    # body acceleration
    filenames += ['body_acc_x_'+group+'.txt', 'body_acc_y_'+group+'.txt',
'body_acc_z_'+group+'.txt']
    # body gyroscope
    filenames += ['body_gyro_x_'+group+'.txt', 'body_gyro_y_'+group+'.txt',
'body_gyro_z_'+group+'.txt']
    # load input data
    X = load_group(filenames, filepath)
    # load class output
    y = load_file(prefix + group + '/y_'+group+'.txt')
    return X, y

# load the dataset, returns train and test X and y elements
def load_dataset(prefix=''):
    # load all train
    trainX, trainy = load_dataset_group('train', prefix + 'HARDataset/')
    print(trainX.shape, trainy.shape)
    # load all test
    testX, testy = load_dataset_group('test', prefix + 'HARDataset/')
    print(testX.shape, testy.shape)
    # zero-offset class values
    trainy = trainy - 1
    testy = testy - 1
    # one hot encode y
    trainy = to_categorical(trainy)
    testy = to_categorical(testy)
    print(trainX.shape, trainy.shape, testX.shape, testy.shape)
    return trainX, trainy, testX, testy

# fit and evaluate a model
def evaluate_model(trainX, trainy, testX, testy):
    verbose, epochs, batch_size = 0, 15, 64
    n_timesteps, n_features, n_outputs = trainX.shape[1], trainX.shape[2], trainy.shape[1]
    model = Sequential()
    model.add(LSTM(100, input_shape=(n_timesteps,n_features)))
    model.add(Dropout(0.5))
    model.add(Dense(100, activation='relu'))
    model.add(Dense(n_outputs, activation='softmax'))
    model.compile(loss='categorical_crossentropy',                    optimizer='adam',
metrics=['accuracy'])
    # fit network
    model.fit(trainX, trainy, epochs=epochs, batch_size=batch_size, verbose=verbose)
    # evaluate model
    _, accuracy = model.evaluate(testX, testy, batch_size=batch_size, verbose=0)
    return accuracy

# summarize scores
def summarize_results(scores):
    print(scores)
    m, s = mean(scores), std(scores)
    print('Accuracy: %.3f%% (+/-%.3f)' % (m, s))

# run an experiment
def run_experiment(repeats=10):
    # load data
    trainX, trainy, testX, testy = load_dataset()
    # repeat experiment
    scores = list()
    for r in range(repeats):
            score = evaluate_model(trainX, trainy, testX, testy)
            score = score * 100.0
            print('>#%d: %.3f' % (r+1, score))
            scores.append(score)
    # summarize results
    summarize_results(scores)

# run the experiment
run_experiment()
```

classify_image.py

```
# USAGE
# python classify_image.py --image images/soccer_ball.jpg --model vgg16

# import the necessary packages
```

```
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.applications import InceptionV3
from tensorflow.keras.applications import Xception # TensorFlow ONLY
from tensorflow.keras.applications import VGG16
from tensorflow.keras.applications import VGG19
#from tensorflow.keras.applications import imagenet_utils
#from keras.applications.vgg16 import preprocess_input, decode_prediction
from keras.applications import imagenet_utils
from tensorflow.keras.applications.inception_v3 import preprocess_input
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.preprocessing.image import load_img
import numpy as np
import argparse
import cv2

# construct the argument parse and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-i", "--image", required=True,
    help="path to the input image")
ap.add_argument("-model", "--model", type=str, default="vgg16",
    help="name of pre-trained network to use")
args = vars(ap.parse_args())

# define a dictionary that maps model names to their classes
# inside Keras
MODELS = {
    "vgg16": VGG16,
    "vgg19": VGG19,
    "inception": InceptionV3,
    "xception": Xception, # TensorFlow ONLY
    "resnet": ResNet50
}

# esnure a valid model name was supplied via command line argument
if args["model"] not in MODELS.keys():
    raise AssertionError("The --model command line argument should "
            "be a key in the `MODELS` dictionary")

# initialize the input image shape (224x224 pixels) along with
# the pre-processing function (this might need to be changed
# based on which model we use to classify our image)
inputShape = (224, 224)
preprocess = imagenet_utils.preprocess_input

# if we are using the InceptionV3 or Xception networks, then we
# need to set the input shape to (299x299) [rather than (224x224)]
# and use a different image pre-processing function
if args["model"] in ("inception", "xception"):
    inputShape = (299, 299)
    preprocess = preprocess_input
```

```
# load our the network weights from disk (NOTE: if this is the
# first time you are running this script for a given network, the
# weights will need to be downloaded first -- depending on which
# network you are using, the weights can be 90-575MB, so be
# patient; the weights will be cached and subsequent runs of this
# script will be *much* faster)
print("[INFO] loading {}...".format(args["model"]))
Network = MODELS[args["model"]]
model = Network(weights="imagenet")

# load the input image using the Keras helper utility while ensuring
# the image is resized to `inputShape`, the required input dimensions
# for the ImageNet pre-trained network
print("[INFO] loading and pre-processing image...")
image = load_img(args["image"], target_size=inputShape)
image = img_to_array(image)

# our input image is now represented as a NumPy array of shape
# (inputShape[0], inputShape[1], 3) however we need to expand the
# dimension by making the shape (1, inputShape[0], inputShape[1], 3)
# so we can pass it through the network
image = np.expand_dims(image, axis=0)

# pre-process the image using the appropriate function based on the
# model that has been loaded (i.e., mean subtraction, scaling, etc.)
image = preprocess(image)

# classify the image
print("[INFO] classifying image with '{}'...".format(args["model"]))
preds = model.predict(image)
print("Type is ",type(model))
P = imagenet_utils.decode_predictions(preds)

# loop over the predictions and display the rank-5 predictions +
# probabilities to our terminal
for (i, (imagenetID, label, prob)) in enumerate(P[0]):
    print("{}. {}: {:.2f}%".format(i + 1, label, prob * 100))

# load the image via OpenCV, draw the top prediction on the image,
# and display the image to our screen
orig = cv2.imread(args["image"])
(imagenetID, label, prob) = P[0][0]
cv2.putText(orig, "Label: {}, {:.2f}%".format(label, prob * 100),
    (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0, 0, 255), 2)
cv2.imshow("Classification", orig)
cv2.waitKey(0)
```

dp_Activity.py

```
# USAGE
# python human_activity_reco.py --model resnet-34_kinetics.onnx --classes action_recognition_kinetics.txt --input example_activities.mp4
# python human_activity_reco.py --model resnet-34_kinetics.onnx --classes
action_recognition_kinetics.txt

# import the necessary packages
import numpy as np
import argparse
import imutils
import sys
import cv2

# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-m", "--model", required=True,
    help="path to trained human activity recognition model")
ap.add_argument("-c", "--classes", required=True,
    help="path to class labels file")
ap.add_argument("-i", "--input", type=str, default="",
    help="optional path to video file")
args = vars(ap.parse_args())

# load the contents of the class labels file, then define the sample
# duration (i.e., # of frames for classification) and sample size
# (i.e., the spatial dimensions of the frame)
CLASSES = open(args["classes"]).read().strip().split("\n")
SAMPLE_DURATION = 16
SAMPLE_SIZE = 112

# load the human activity recognition model
print("[INFO] loading human activity recognition model...")
net = cv2.dnn.readNet(args["model"])

# grab a pointer to the input video stream
print("[INFO] accessing video stream...")
vs = cv2.VideoCapture(args["input"] if args["input"] else 0)

# loop until we explicitly break from it
while True:
    # initialize the batch of frames that will be passed through the
    # model
    frames = []

    # loop over the number of required sample frames
    for i in range(0, SAMPLE_DURATION):
        # read a frame from the video stream
        (grabbed, frame) = vs.read()

        # if the frame was not grabbed then we've reached the end of
        # the video stream so exit the script
        if not grabbed:
            print("[INFO] no frame read from stream - exiting")
            sys.exit(0)
```

```
		# otherwise, the frame was read so resize it and add it to
		# our frames list
		frame = imutils.resize(frame, width=400)
		frames.append(frame)

	# now that our frames array is filled we can construct our blob
	blob = cv2.dnn.blobFromImages(frames, 1.0,
		(SAMPLE_SIZE, SAMPLE_SIZE), (114.7748, 107.7354, 99.4750),
		swapRB=True, crop=True)
	blob = np.transpose(blob, (1, 0, 2, 3))
	blob = np.expand_dims(blob, axis=0)

	# pass the blob through the network to obtain our human activity
	# recognition predictions
	net.setInput(blob)
	outputs = net.forward()
	label = CLASSES[np.argmax(outputs)]

	# loop over our frames
	for frame in frames:
		# draw the predicted activity on the frame
		cv2.rectangle(frame, (0, 0), (300, 40), (0, 0, 0), -1)
		cv2.putText(frame, label, (10, 25), cv2.FONT_HERSHEY_SIMPLEX,
			0.8, (255, 255, 255), 2)

		# display the frame to our screen
		cv2.imshow("Activity Recognition", frame)
		key = cv2.waitKey(1) & 0xFF

		# if the `q` key was pressed, break from the loop
		if key == ord("q"):
			break
```

human_activity_reco.py

```
# USAGE
# python human_activity_reco.py --model resnet-34_kinetics.onnx --classes action_recognition_kinetics.txt --input example_activities.mp4
# python human_activity_reco.py --model resnet-34_kinetics.onnx --classes action_recognition_kinetics.txt

# import the necessary packages
import numpy as np
import argparse
import imutils
import sys
import cv2

# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
```

```
ap.add_argument("-m", "--model", required=True,
    help="path to trained human activity recognition model")
ap.add_argument("-c", "--classes", required=True,
    help="path to class labels file")
ap.add_argument("-i", "--input", type=str, default="",
    help="optional path to video file")
args = vars(ap.parse_args())

# load the contents of the class labels file, then define the sample
# duration (i.e., # of frames for classification) and sample size
# (i.e., the spatial dimensions of the frame)
CLASSES = open(args["classes"]).read().strip().split("\n")
SAMPLE_DURATION = 16
SAMPLE_SIZE = 112

# load the human activity recognition model
print("[INFO] loading human activity recognition model...")
net = cv2.dnn.readNet(args["model"])

# grab a pointer to the input video stream
print("[INFO] accessing video stream...")
vs = cv2.VideoCapture(args["input"] if args["input"] else 0)

# loop until we explicitly break from it
while True:
    # initialize the batch of frames that will be passed through the
    # model
    frames = []

    # loop over the number of required sample frames
    for i in range(0, SAMPLE_DURATION):
            # read a frame from the video stream
            (grabbed, frame) = vs.read()

            # if the frame was not grabbed then we've reached the end of
            # the video stream so exit the script
            if not grabbed:
                    print("[INFO] no frame read from stream - exiting")
                    sys.exit(0)

            # otherwise, the frame was read so resize it and add it to
            # our frames list
            frame = imutils.resize(frame, width=400)
            frames.append(frame)

    # now that our frames array is filled we can construct our blob
    blob = cv2.dnn.blobFromImages(frames, 1.0,
            (SAMPLE_SIZE, SAMPLE_SIZE), (114.7748, 107.7354, 99.4750),
            swapRB=True, crop=True)
    blob = np.transpose(blob, (1, 0, 2, 3))
    blob = np.expand_dims(blob, axis=0)
    # pass the blob through the network to obtain our human activity
    # recognition predictions
    net.setInput(blob)
    outputs = net.forward()
    label = CLASSES[np.argmax(outputs)]

    # loop over our frames
    for frame in frames:
            # draw the predicted activity on the frame
            cv2.rectangle(frame, (0, 0), (300, 40), (0, 0, 0), -1)
            cv2.putText(frame, label, (10, 25), cv2.FONT_HERSHEY_SIMPLEX,
                    0.8, (255, 255, 255), 2)

            # display the frame to our screen
            cv2.imshow("Activity Recognition", frame)
            key = cv2.waitKey(1) & 0xFF

            # if the `q` key was pressed, break from the loop
            if key == ord("q"):
                    break
```

mainRun.py

```
import cv2
import argparse
import os
import subprocess
from random import randrange

folder = 'frames'
for filename in os.listdir(folder):
   file_path = os.path.join(folder, filename)
   try:
      if os.path.isfile(file_path) or os.path.islink(file_path):
         os.unlink(file_path)
      elif os.path.isdir(file_path):
         shutil.rmtree(file_path)
   except Exception as e:
      print('Failed to delete %s. Reason: %s' % (file_path, e))

ap = argparse.ArgumentParser()
ap.add_argument("-i", "--input", type=str, default="",
    help="optional path to video file")
args = vars(ap.parse_args())

def FrameCapture(path):
   vidObj = cv2.VideoCapture(path)
   count = 0
   success = 1
   #while success:
   while count<=25:
        # vidObj object calls read
        # function extract frames
        success, image = vidObj.read()

        # Saves the frames with frame-count
        cv2.imwrite("frames/frame%d.jpg" % count, image)

        count += 1

# Driver Code
if __name___== '__main__':
    # Calling the function
    FrameCapture(args["input"])
    inputImage = "frames/frame"+str(randrange(25))+".jpg"
    runvalue = "classify_image.py -i "+inputImage
    subprocess.call("python "+runvalue)
    harActivity = "human_activity_reco.py --model resnet-34_kinetics.onnx --classes
action_recognition_kinetics.txt --input "+args["input"]
    subprocess.call("python "+harActivity)
    scores = "Reportgenearation.py"
    subprocess.call("python "+scores)
```

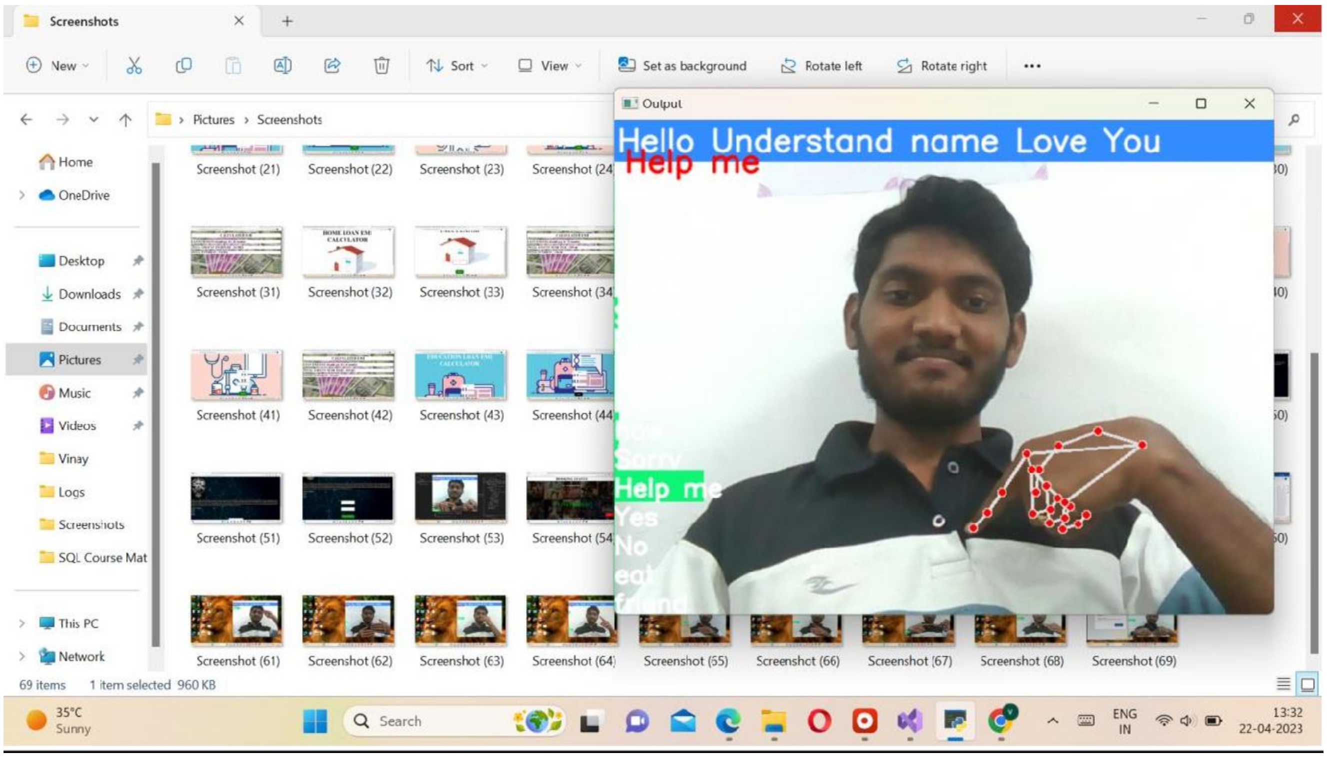
Reportgeneration.py

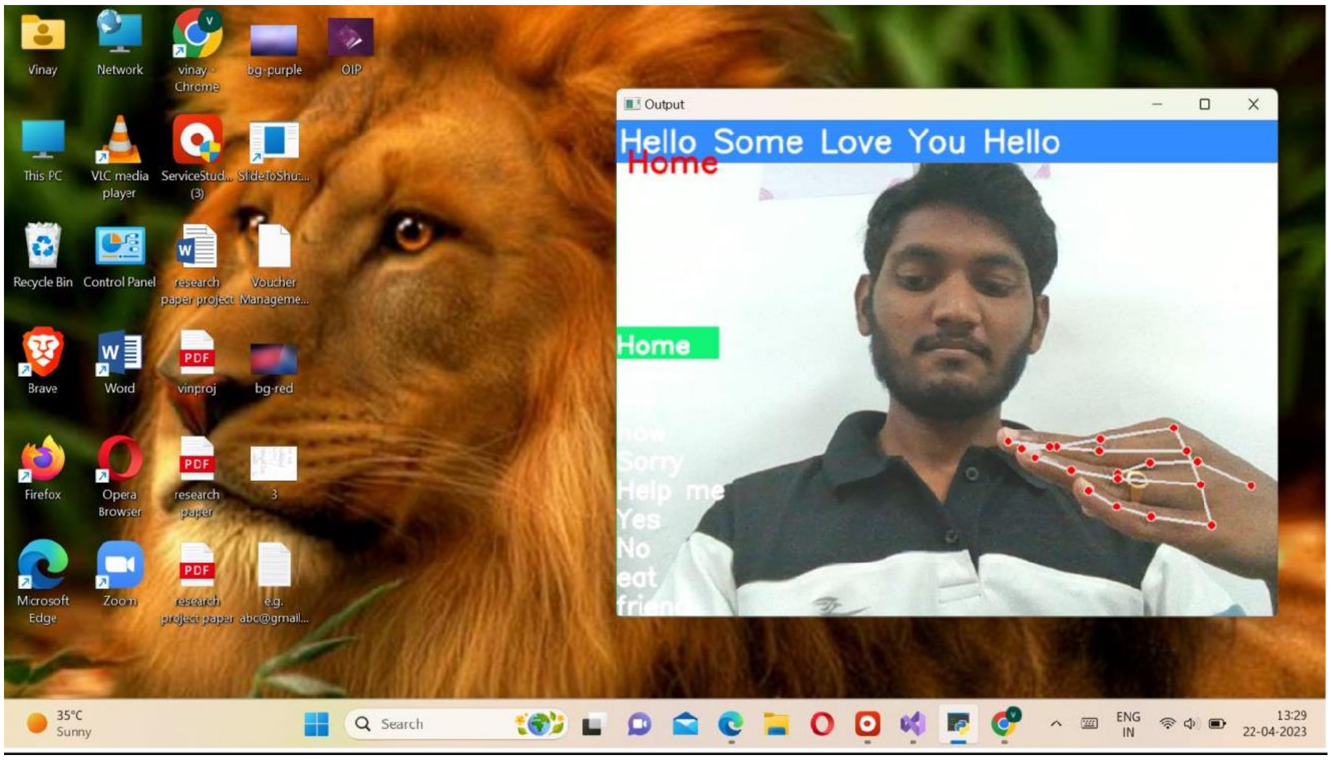
```
# demonstration of calculating metrics for a neural network model using sklearn
from sklearn.datasets import make_circles
from sklearn.datasets import load_sample_image
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import cohen_kappa_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import confusion_matrix
from keras.models import Sequential
from keras.layers import Dense
from sklearn.feature_extraction import image
import seaborn as sns
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
import matplotlib.pyplot as plt

# generate and prepare the dataset
def get_data():
        # generate dataset
        X, y = make_circles(n_samples=1000, noise=0.1, random_state=1)
        #print("Which type you are ..?= am ",len(X),len(y))
        #print(X)
        #print("..........................")
        #print(y)
        # split into train and test
        n_test = 500
```

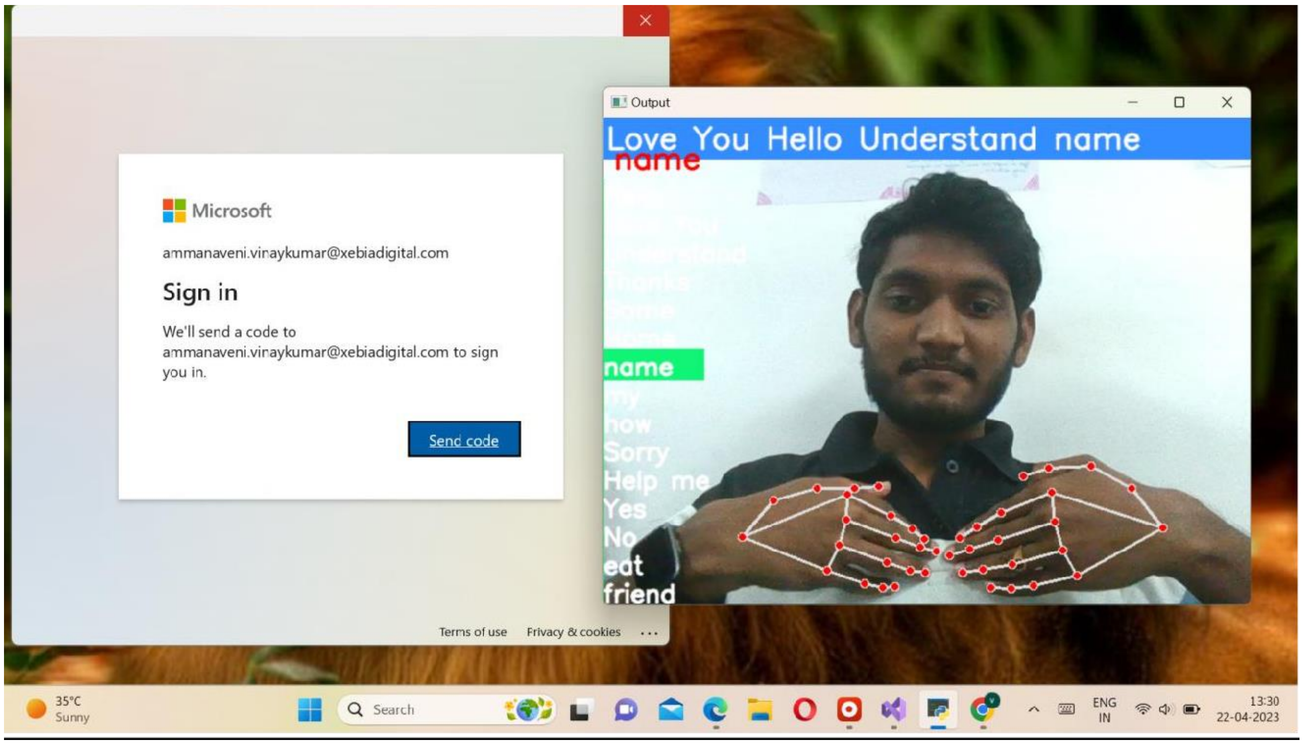
```
    trainX, testX = X[:n_test, :], X[n_test:, :]
    trainy, testy = y[:n_test], y[n_test:]
    one_image = load_sample_image("flower.jpg")
    print('Image shape: {}'.format(one_image.shape))
    patches = image.extract_patches_2d(one_image, (2, 2))
    print('Patches shape: {}'.format(patches.shape))
    print(patches[1])
    print(patches[800])
    return trainX, trainy, testX, testy

# define and fit the model
def get_model(trainX, trainy):
    # define model
    model = Sequential()
    model.add(Dense(100, input_dim=2, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    # compile model
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    # fit model
    model.fit(trainX, trainy, epochs=300, verbose=0)
    return model

# generate data
trainX, trainy, testX, testy = get_data()
# fit model
model = get_model(trainX, trainy)


# predict probabilities for test set
yhat_probs = model.predict(testX, verbose=0)
# predict crisp classes for test set
yhat_classes = model.predict_classes(testX, verbose=0)
# reduce to 1d array
yhat_probs = yhat_probs[:, 0]
yhat_classes = yhat_classes[:, 0]

# accuracy: (tp + tn) / (p + n)
accuracy = accuracy_score(testy, yhat_classes)
print('Accuracy: %f' % accuracy)
# precision tp / (tp + fp)
precision = precision_score(testy, yhat_classes)
print('Precision: %f' % precision)
# recall: tp / (tp + fn)
recall = recall_score(testy, yhat_classes)
print('Recall: %f' % recall)
# f1: 2 tp / (2 tp + fp + fn)
f1 = f1_score(testy, yhat_classes)
print('F1 score: %f' % f1)

# kappa
kappa = cohen_kappa_score(testy, yhat_classes)
```

```
print('Cohens kappa: %f' % kappa)
# ROC AUC
auc = roc_auc_score(testy, yhat_probs)
print('ROC AUC: %f' % auc)
# confusion matrix
f,ax = plt.subplots(figsize=(8, 8))
matrix = confusion_matrix(testy, yhat_classes)
sns.heatmap(matrix, annot=True, linewidths=0.01,cmap="Blues",linecolor="gray", fmt=
'.1f',ax=ax)
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()
print(matrix)
```

B) SCREENSHORT







Microsoft

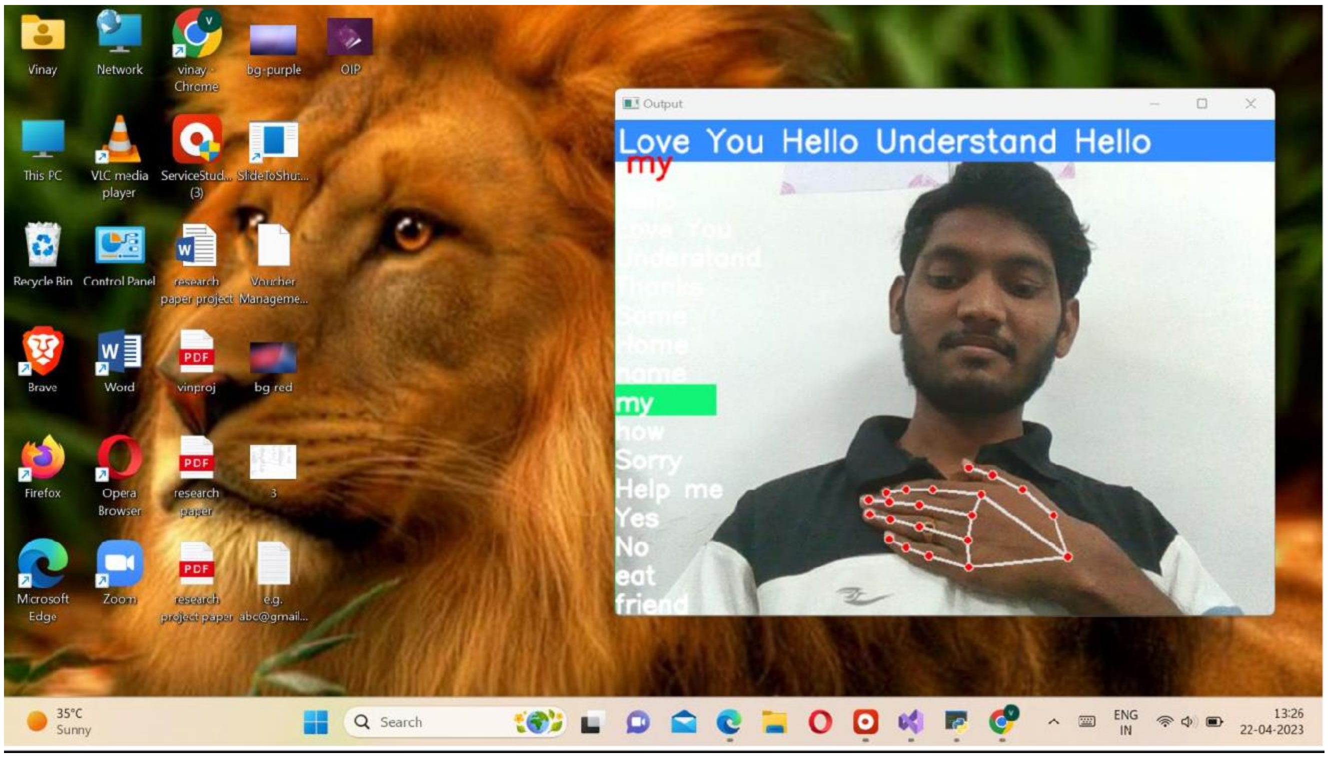
ammanaveni.vinaykumar@xebiadigital.com

Sign in

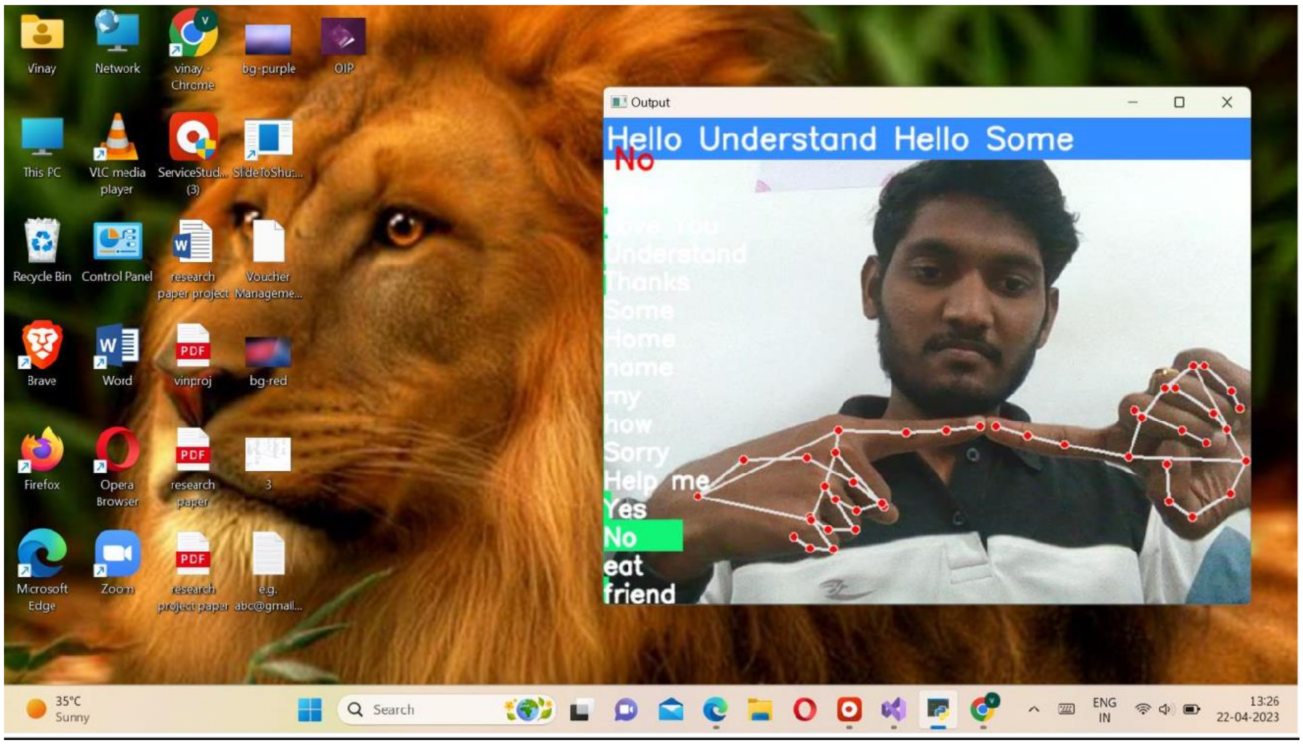
We'll send a code to ammanaveni.vinaykumar@xebiadigital.com to sign you in.

Send code

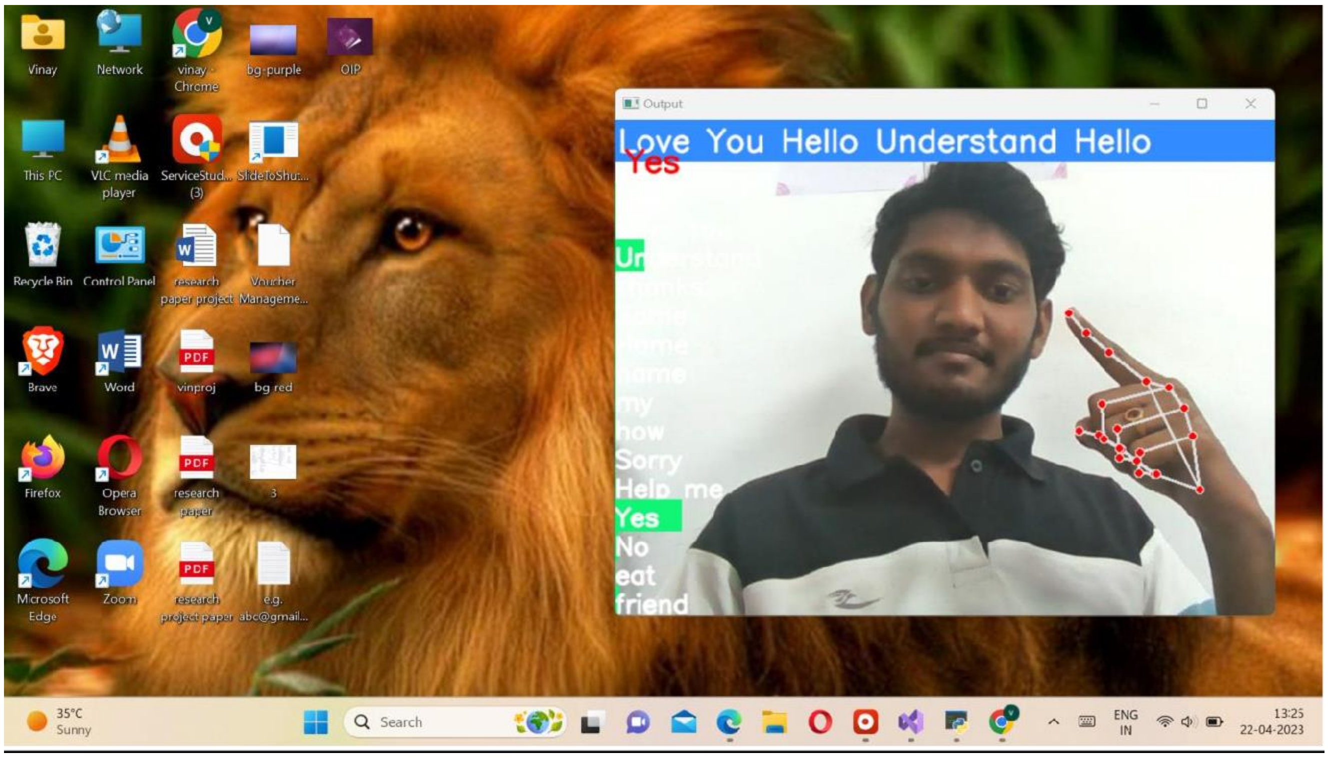
Terms of use Privacy & cookies ...

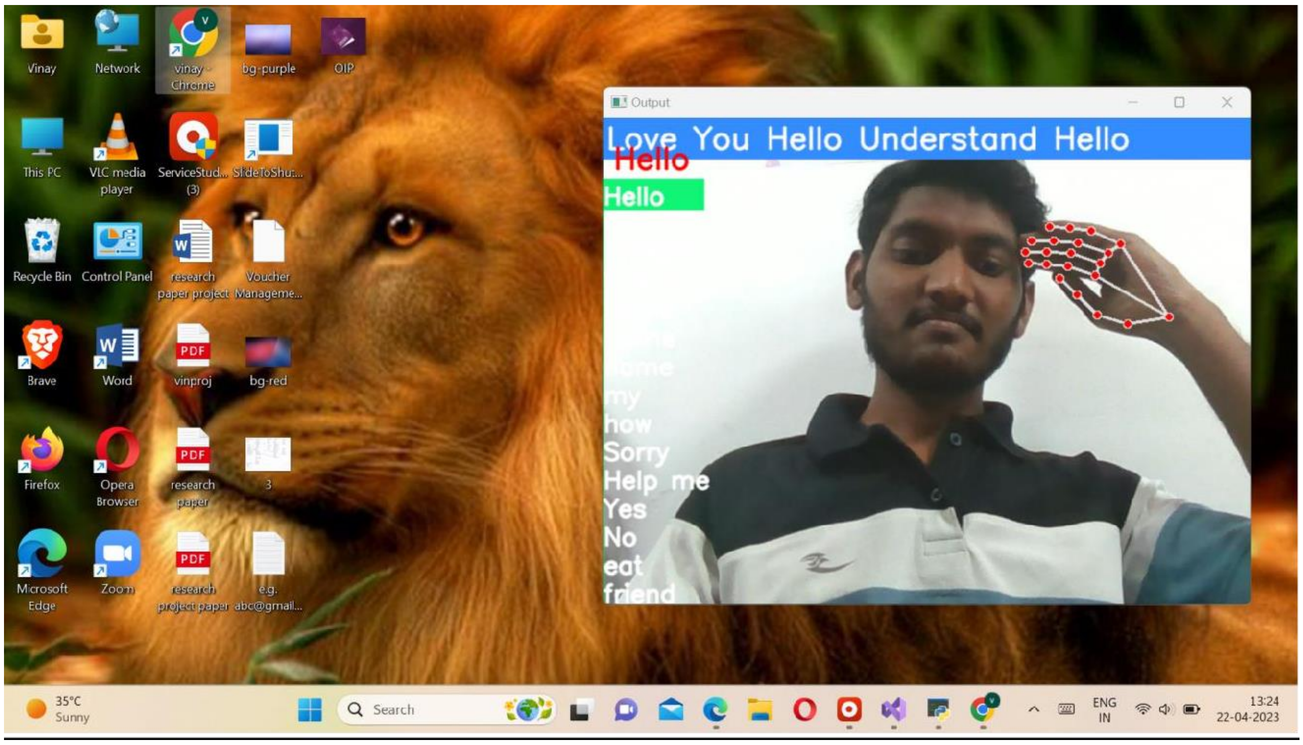


```
Love You Hello Understand Hello
my
Hello
Love You
Understand
Thanks
Some
Home
name
my
how
Sorry
Help me
Yes
No
eat
friend
```



```
Hello Understand Hello Some
No
Love You
Understand
Thanks
Some
Home
name
my
how
Sorry
Help me
Yes
No
eat
friend
```


Output
Love You Hello Understand Hello
Yes
Understand
Thanks
Name
Home
name
my
how
Sorry
Help me
Yes
No
eat
friend


Output
Love You Hello Understand Hello
Hello
Hello
name
home
my
how
Sorry
Help me
Yes
No
eat
friend