**CHAPTER 1**

# INTRODUCTION

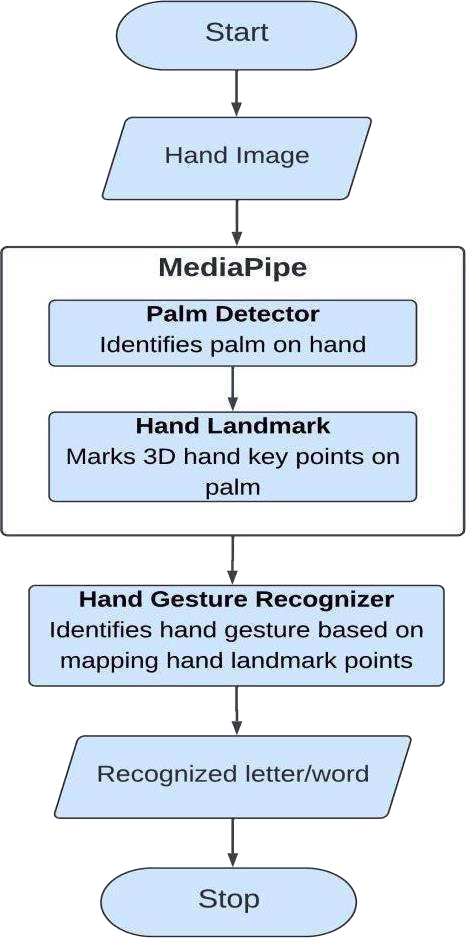
## 1.1 INTRODUCTION

1. Humans communicate with one another using natural language channels such as words and writing, or by body language (gestures) such as hand motions, head gestures, facial expression, lip motion, and so forth. Comprehending sign language is equally as vital as understanding natural language [13].
2. People with hearing impairment use sign language as their preferred mode of communication. Without a translation, people with hearing impairments have difficulty speaking with other hearing people. As a result, implementing a system that understands sign language would have a substantial positive impact on the social lives of deaf people. According to the World Health Organization, 466 million individuals worldwide (more than 5 percent of the population) have impaired hearing, with 34 million of them being teens (WHO). According to studies, by 2050, these numbers will have surpassed 900 million. Furthermore, the majority of cases of profound hearing loss, which afflict millions of individuals, occur in low and middle-income nations [2].
3. Furthermore, the majority of cases of substantial hearing loss, which affects millions of individuals, occur in low- and middle-income nations. There are more than 135 distinct sign languages spoken worldwide, including American Sign Language (ASL), British Sign Language (BSL), and Indian Sign Language (ISL) [15].
4. Machine learning enables the development of systems that accurately interpret sign language, which can greatly improve communication and social lives of deaf people. These technologies are particularly important for those living in low and middle-income nations where the majority of hearing impairments occur. The growing prevalence of hearing loss worldwide highlights the urgent need for technological solutions to help bridge the communication gap between hearing-impaired individuals and the rest of society.
5. Machine learning is a branch of artificial intelligence that deals with the methods that let computers extract meaning from data and create AI applications. In the meanwhile, deep learning is a subset of machine learning that enables computers to resolve increasingly challenging issues [11]. As deep learning develops transferable answers, it is more powerful than traditional machine learning. Through neural networks, or layers of neurons/units, deep learning algorithms are able to produce transferable solutions [12]. Deep learning is a subset of machine learning where a computer program learns to carry out classification operations on complex input such as images, text, or sound. These algorithms are able to execute at a state-of-the-art (SOTA) level of accuracy and, in certain situations, even surpass humans. Numerous labeled data points and intricate neural network topologies are used to learn them. It is a vital part of modern innovations like self-driving cars, virtual assistants, and face recognition.
6. In our research, we have thoroughly examined the existing literature on Sign language recognition. We will now focus on the most notable research papers and discuss their methods for feature extraction, image pre- processing, and image classification, which employ a variety of algorithms including SVM, KNN, and CNN. Additionally, we have examined several image-processing techniques, including Canny-edge detection, Convexhull algorithm, and Gaussian blur filter, among others.
7. A Microsoft Kinect camera was used to create a sign language recognition system in [6]. This was chosen to allow the whole programme to be independent of restrictions such as poor illumination, loud input, and so on. Depth and Motion were the two main feature capturing modules used in their methodology. In fact, a feature vector was calculated for each frame of the video series, and some preprocessing was applied to each frame to eliminate undesired noise and provide a clean image of the depth map. They used the Gaussian blur filter 15 and the Erosion filter to do this and also presented the depth information using a 256-bin histogram for a depth image. They were able to create the feature matrix for that particular video sequence or gesture using the combined array of feature vectors from all of the frames in the video sequence. Following this preprocessing, the feature matrix was given as an input to a multi-class SVM classifier to construct an appropriate Machine Learning model for classification of the test files using kernel functions, with the linear and RBF kernels being specifically employed. The total accuracy achieved was between 81.48 and 87.67 percent. However, this work was unable to investigate other high-level characteristics such as optical flow information, motion gradient information, and so on, which may have improved accuracy performance.
8. A more precise real-time Hand Gesture Recognition (HGR) system based on American Sign Language is the primary goal of [8], which is to illustrate (ASL). The combination of Kcurvature and convex hull approaches is proposed as a novel feature extraction technique. This method, known as the "K Convex Hull" technique, can recognize fingers with extreme precision. An ANN is used in this system together with feed forward and reverse propagation techniques to train a network with 30 feature vectors to accurately identify 37 indications of American alphanumeric letters, which is beneficial for HCI applications. The entire gesture recognition rate of this system in a real-time scenario is

94.32 percent.

1. The study described in [17] involved the use of a camera to capture images of Indian Sign Language (ISL) hand gestures. Before feature extraction from image, its pre- processing was done. In this work, a unique approach of the Canny Edge 16 Detection Algorithm was discussed. It was found that Canny edge detection algorithm is able to detect both strong and weak edges proving it to be more accurate than other techniques like Laplacian or Gaussian. Once the image's necessary elements have been extracted.

## 1.2 BLOCK DIAGRAM



**Fig:-1 Block Diagram**

## 1.3 DESCRIPTION OF BLOCK DIAGRAM

Sure, here are some key points about MediaPipe's involvement in hand gesture recognition:

Introduction of MediaPipe\*\*: MediaPipe is an open-source framework developed by Google that offers a comprehensive solution for building perception pipelines. It provides prebuilt yet customizable components for various perceptual tasks, including hand gesture recognition.

Hand Tracking Module\*\*: MediaPipe includes a robust hand tracking module that can detect and track the movements of a hand in real-time from a video feed. This module utilizes machine learning models to accurately estimate the 3D position and orientation of the hand.

1. \*\*Gesture Recognition Pipeline\*\*: MediaPipe extends its capabilities beyond hand tracking to include gesture recognition. By integrating machine learning models trained on vast datasets, MediaPipe can classify specific hand gestures based on the tracked hand's movements and positions.

1. Customizable Gesture Recognition\*\*: One of the strengths of MediaPipe is its flexibility. Developers can train their own gesture recognition models using their datasets and integrate them seamlessly into the MediaPipe framework. This allows for the recognition of custom gestures tailored to specific applications or user needs.

1. \*\*Real-Time Performance\*\*: MediaPipe is designed for real-time performance, making it suitable for applications where low-latency gesture recognition is crucial, such as augmented reality (AR), virtual reality (VR), or interactive user interfaces.

1. \*\*Cross-Platform Support\*\*: MediaPipe offers support for various platforms, including mobile devices, desktops, and embedded systems. This cross-platform compatibility ensures that applications built using MediaPipe can reach a wide range of users across different devices.

1. \*\*Integration with Other MediaPipe Components\*\*: MediaPipe's hand gesture recognition capabilities can be seamlessly integrated with other components within the MediaPipe framework, such as face detection, object detection, or pose estimation. This allows developers to build complex multi-modal perception pipelines efficiently.

Overall, MediaPipe's involvement in hand gesture recognition showcases its versatility and effectiveness in enabling developers to build robust and efficient perceptual applications with minimal effort**.**

**1.4 Additional Features**

**Hand Tracking:**

The HandDetector class from the cvzone.HandTrackingModule is used to detect hands in real-time video frames.

The maxHands parameter is set to 1, indicating that the system tracks a single hand.

The findHands method returns information about detected hands, including bounding boxes and landmarks (key points).

**Image Preprocessing:**

The code captures video frames from the webcam using cv2.VideoCapture(0).

For each detected hand, it crops the hand region based on the bounding box.

The cropped image is resized to a fixed size (e.g., 300x300 pixels).

The resized hand image is placed on a white canvas (imgWhite) for further processing.

**Aspect Ratio Handling:**

The system adjusts the cropped image based on the aspect ratio (height-to-width ratio) of the hand.

If the aspect ratio is greater than 1, the image is resized to fit the canvas width.

Otherwise, it is resized to fit the canvas height.

**Saving Images:**

When the user presses the “s” key, the system saves the processed hand image to a specified folder.

The filename includes a timestamp to ensure uniqueness.

**Visual Feedback:**

The system displays the original video frame (img), the cropped hand region (imgCrop), and the resized hand image on a white canvas (imgWhite).

**Counter:**

The counter variable keeps track of the number of saved hand images.

**CHAPTER 2**

**LITERATURE SURVEY**

A literature survey on hand gesture recognition typically covers a range of topics, including techniques, algorithms, datasets, challenges, and applications. Here's an overview of what such a survey might include:

Introduction to Hand Gesture Recognition: This section provides an overview of hand gesture recognition, its importance, and its applications in various fields such as humancomputer interaction, sign language recognition, robotics, and augmented reality.

Hand Gesture Representation: Discusses different methods for representing hand gestures, including 2D and 3D representations, skeletal models, depth maps, and image-based representations.

Feature Extraction: Covers techniques for extracting features from hand gesture data, including traditional methods like Histogram of Oriented Gradients (HOG), Haar-like features, as well as deep learning-based feature extraction methods such as Convolutional Neural Networks (CNNs).

Gesture Recognition Algorithms: Reviews different machine learning and deep learning algorithms used for gesture recognition, including Hidden Markov Models (HMMs), Support Vector Machines (SVMs), Recurrent Neural Networks (RNNs), Convolutional Neural Networks (CNNs), and their variants.

Datasets: Surveys publicly available datasets for hand gesture recognition, including benchmark datasets such as ChaLearn Looking at People (LAP) and American Sign Language (ASL) datasets. Discusses their characteristics, size, annotations, and suitability for different tasks.

Challenges and Limitations: Identifies challenges and limitations in hand gesture recognition, such as occlusion, viewpoint variation, lighting conditions, background clutter, and intra-class variation. Discusses how different techniques address these challenges.

Recent Advances and Trends: Highlights recent advances in hand gesture recognition research, including novel algorithms, architectures, and datasets. Discusses emerging trends such as multimodal fusion, self-supervised learning, and domain adaptation.

Applications: Discusses various applications of hand gesture recognition, including gesture-based interaction systems, sign language translation, virtual reality, healthcare, robotics, and automotive safety.

Evaluation Metrics: Covers evaluation metrics commonly used for assessing the performance of hand gesture recognition systems, including accuracy, precision, recall, F1-score, and mean average precision (mAP).

Conclusion and Future Directions: Summarizes the key findings of the literature survey and identifies potential future research directions in hand gesture recognition, such as improving robustness, scalability, and real-time performance, as well as exploring new application domains.

A comprehensive literature survey on hand gesture recognition would delve into each of these topics, providing a thorough understanding of the state-of-the-art techniques, challenges, and applications in the field.

## 

## 2.1 Dataset

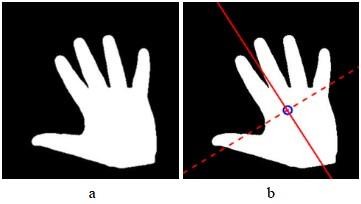
In this work, we have utilized the ASL dataset [20] consisting of 51 classes, with approximately 4000 images per class. The classes comprise the alphabet, numbers, and commonly used words such as ‘Hello’, ‘Help’, and ‘Stop’. The alphabet class enables the formation of new words through fingerspelling, where individual letters are used to represent words without a designated sign symbol.

A Python script was employed to efficiently convert the image class folders into a .csv file, which stores the (x, y, z) coordinates of all landmark points of each sign with their respective outputs. An 80:20 train-test split was implemented to improve the model's feature extraction process.



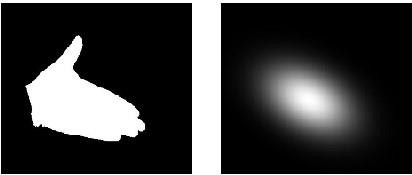
**Fig. 2.1 Dataset**

Hasan applied multivariate Gaussian distribution to recognize hand gestures using non geometric features. The input hand image is segmented using two different methods; skin color based segmentation by applying HSV color model and clustering based thresholding techniques. Some operations are performed to capture the shape of the hand to extract hand feature; the modified Direction Analysis Algorithm are adopted to find a relationship between statistical parameters (variance and covariance) from the data, and used to compute object (hand) slope and trend by finding the direction of the hand gesture , As shown in the below Figure



**Fig:-2.2 Variance and Co Variance**

Then Gaussian distinction is applied on the segmented image, and it takes the direction of the hand as shown in figure 1.4.

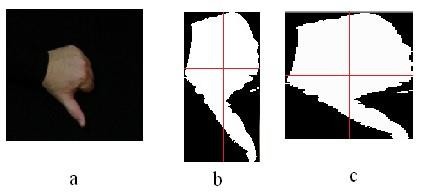


**Fig:-2.3 Gaussians Distinction**

The input image are converted into HSV color model, resized into 80x64 and some image preprocessing operations are applied to segment the hand [31]from a uniform background , features are extracted using histogram technique and Hough algorithm. Feed forward Neural Networks with three layers are used for gesture classification. 8 samples are used for each 26 characters in sign language, for each gesture, 5 samples are used for training and 3samples for testing, the system achieved 92.78% recognition rate using MATLAB language

Two methods are used for extraction the features; firstly by using the edge mages,

and secondly by using normalized features where only the brightness values of pixels are calculated and other black pixels are neglected to reduce the length of the feature vector [5]. The database consists of 6 different gestures, 10 samples per gesture are used, 5 samples for training and 5 samples for testing. The recognition rate for the normalized feature problem achieved better performance than the normal feature method, 95% recognition rate for the former method and 84% for the latter one



**Fig:-2.4 Gesture Accuracy**

* **Human-computer interaction:** Gesture recognition can be used to control computers, smartphones, and other devices through gestures, such as swiping, tapping, and pinching.
* **Gaming:** Gesture recognition can be used to control characters and objects in video games, making the gaming experience more immersive and interactive.
* **Virtual and augmented reality:** Gesture recognition can be used to interact with virtual and augmented reality environments, allowing users to control and manipulate objects in those environments.
* **Robotics:** Gesture recognition can be used to control robots, allowing them to perform tasks based on the user’s gestures.
* **Sign language recognition:** Gesture recognition can be used to recognize and translate sign language into spoken or written language, helping people who are deaf or hard of hearing communicate with others.
* **Automotive:** Gesture recognition can be used in cars to control various functions such as radio, AC, and navigation systems.
* **Healthcare:** Gesture recognition can be used in rehabilitation of patients with physical disabilities.

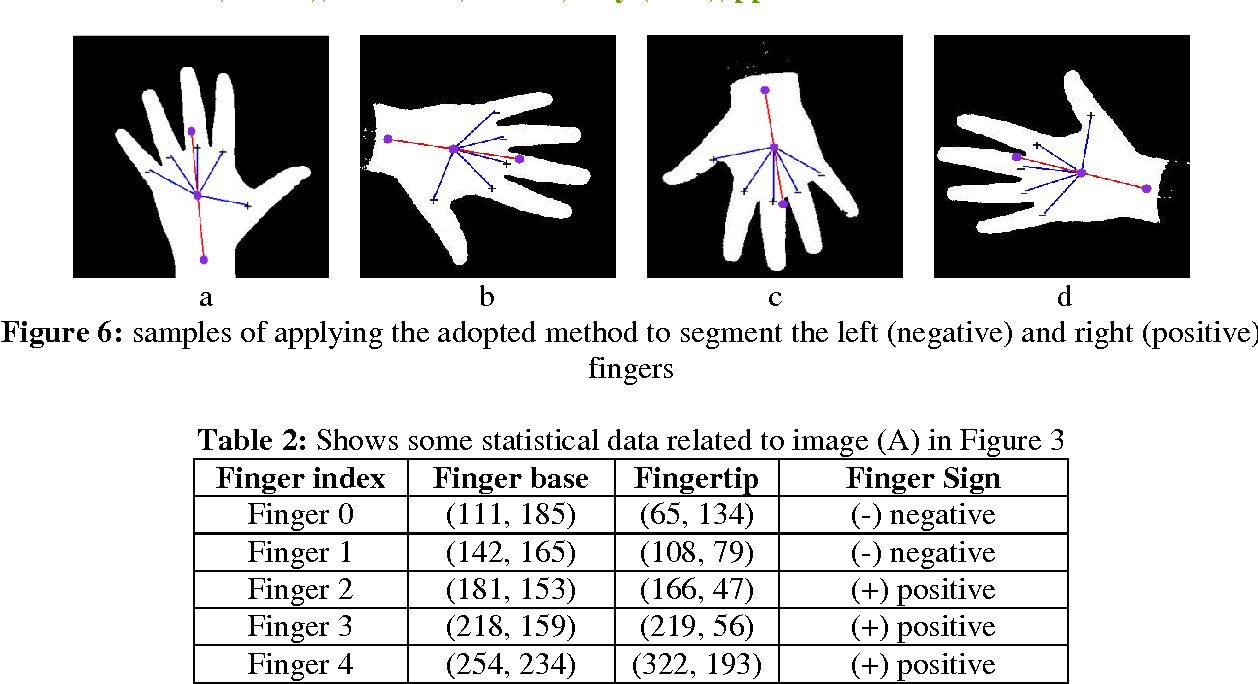
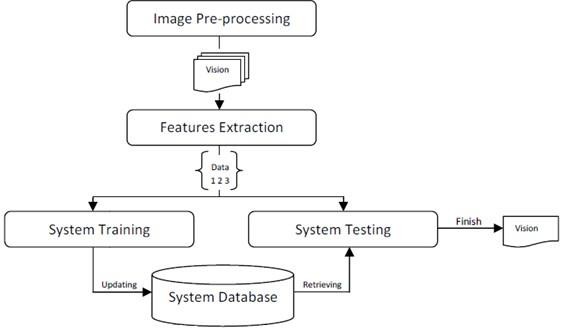


Fig:-2.5 Samples of Applying Adopted Method to Images

**CHAPTER 3**

# PROJECT DESCRIPTION

## 3.1 BLOCK DIAGRAM OF ENTIRE SYSTEM



**Fig:- 3 Block diagram of System**

### 3.1.2 APPLICATION AREAS OF HAND GESTURES SYSTEM

Hand gestures recognition system has been applied for different applications on different domains, as mentioned in [7][9] including; sign language translation, virtual environments, smart surveillance, robot control, medical systems etc. overview of some hand gesture application areas are listed below[7][8].

*A.* Sign Language Recognition:

Since the sign language is used for interpreting and explanations of a certain subject during the conversation, it has received special attention [7]. A lot of systems have been proposed to recognize gestures using different types of sign languages [8]. For example [8] recognized American Sign Language ASL using boundary histogram, MLP neural network and dynamic programming matching. [28] recognized Japanese sign language JSL using Recurrent Neural Network, 42 alphabet and 10 words. [25] recognized Arabic Sign language ArSL using two different types of Neural Network, Partially and Fully Recurrent neural Network

*B. Robot Control:*

Controlling the robot using gestures considered as one of the interesting applications in this field [6]. [16] proposed a system that uses the numbering to count the five fingers for controlling a robot using hand pose signs. The orders are given to the robot to perform a particular task [16], where each sign has a specific meaning and represents different function for example, “one” means “move forward”, “five” means “stop”, and so on.

*C.* Graphic Editor Control*:*

Graphic editor control system requires the hand gesture to be tracked and located as a preprocessing operation [7]. [20] used 12 dynamic gestures for drawing and editing graphic system. Shapes for drawing are; triangle, rectangular, circle, arc, horizontal and vertical line for drawing, and commands for editing graphic system are; copy, delete, move, swap, undo, and close [20].

*D.* Virtual Environments ( VEs):

One of the popular applications in gesture recognition system is virtual environments VEs, especially for communication media systems [9]. [29] provided 3D pointing gesture recognition for natural human computer Interaction HCI in a real-time from binocular views. The proposed system is accurate and independent of user characteristics and environmental changes [29].

*E. Numbers Recognition*:

Another recent application of hand gesture is recognizing numbers. [13] proposed an automatic system that could isolate and recognize a meaningful gesture from hand motion of Arabic numbers from 0 to 9 in a real time system using HMM. *F. Television Control:*

Hand postures and gestures are used for controlling the Television device [9]. In [30] a set of hand gesture are used to control the TV activities, such as turning the TV on and off, increasing and decreasing the volume, muting the sound, and changing the channel using open and close hand [30].

*G.* 3D Modeling

To build 3D modeling, a determination of hand shapes are needed to create, built and view 3D shape of the hand [9]. Some systems built the 2D and 3D objects using hand silhouette [9]. 3D hand modeling can be used for this purpose also which still a promising field of research.

**3.2 EXISTING MODEL**

**MEDIAPIPE**

MediaPipe is a customizable machine learning solutions framework developed by Google. It is an open-source and cross-platform framework, and it is very lightweight. MediaPipe comes with some pre-trained ML solutions such as face detection, pose estimation, hand recognition, object detection, etc.

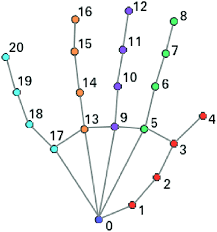


Fig:- 3.1 MediaPipe Module Detection

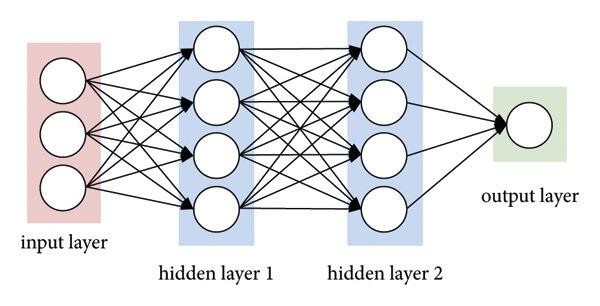


Fig:-3.2 MediaPipe Layers Models

We’ll first use MediaPipe to recognize the hand and the hand key points. MediaPipe returns a total of 21 key points for each detected hand.

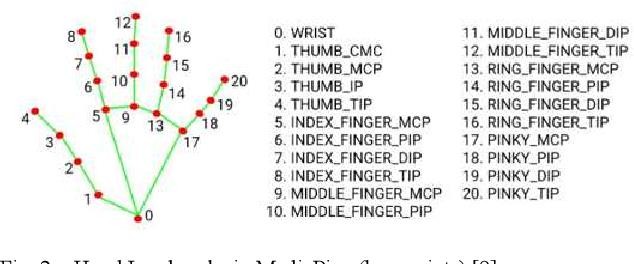


Fig :-3.3 MediaPipe Key Points

These key points will be fed into a pre-trained gesture recognizer network to recognize the hand pose.

**3.3 TENSORFLOW**

TensorFlow is an open-source library for machine learning and deep learning developed by the Google brains team. It can be used across a range of tasks but has a particular focus on deep neural networks. **Neural Networks** are also known as artificial neural networks. It is a subset of machine learning and the heart of deep learning algorithms. The concept of Neural networks is inspired by the human brain. It mimics the way that biological neurons send signals to one another. Neural networks are composed of node layers, containing an input layer, one or more hidden layers, and an output layer.

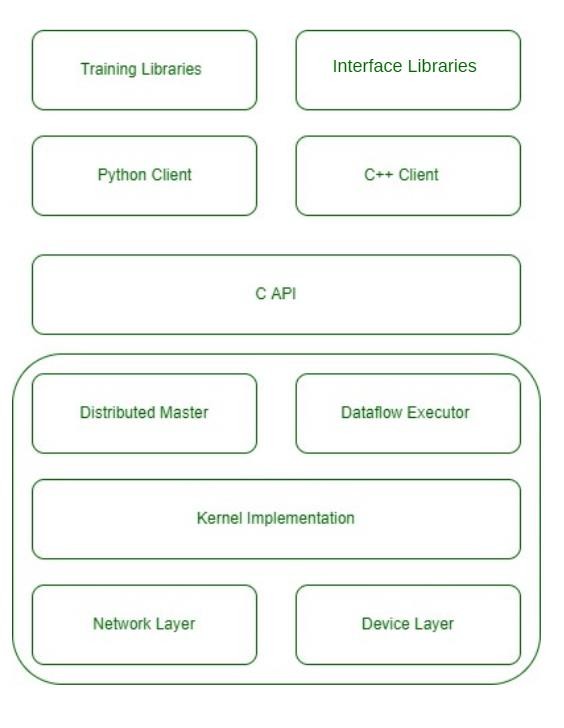


Fig:-3.4 TensorFlow Module

## 

### Tensor processing unit (TPU)[[edit](https://en.wikipedia.org/w/index.php?title=TensorFlow&action=edit&section=4)]

In May 2016, Google announced its [Tensor processing unit](https://en.wikipedia.org/wiki/Tensor_processing_unit) (TPU), an [application-specific integrated circuit](https://en.wikipedia.org/wiki/Application-specific_integrated_circuit) ([ASIC](https://en.wikipedia.org/wiki/Application-specific_integrated_circuit), a hardware chip) built specifically for machine learning and tailored for TensorFlow. A TPU is a programmable [AI accelerator](https://en.wikipedia.org/wiki/AI_accelerator_(computer_hardware)) designed to provide high [throughput](https://en.wikipedia.org/wiki/Throughput) of low-precision [arithmetic](https://en.wikipedia.org/wiki/Arithmetic) (e.g., [8-bit](https://en.wikipedia.org/wiki/8-bit)), and oriented toward using or running models rather than [training](https://en.wikipedia.org/wiki/Supervised_learning) them. Google announced they had been running TPUs inside their data centers for more than a year, and had found them to deliver an [order of magnitude](https://en.wikipedia.org/wiki/Order_of_magnitude) better-optimized [performance per watt](https://en.wikipedia.org/wiki/Performance_per_watt) for machine learning.

In May 2017, Google announced the second-generation, as well as the availability of the TPUs in [Google Compute Engine](https://en.wikipedia.org/wiki/Google_Compute_Engine).[[23]](https://en.wikipedia.org/wiki/TensorFlow#cite_note-23) The second-generation TPUs deliver up to 180 [teraflops](https://en.wikipedia.org/wiki/FLOPS) of performance, and when organized into clusters of 64 TPUs, provide up to 11.5 [petaflops](https://en.wikipedia.org/wiki/FLOPS).[[*citation needed*](https://en.wikipedia.org/wiki/Wikipedia:Citation_needed)]

In May 2018, Google announced the third-generation TPUs delivering up to 420 [teraflops](https://en.wikipedia.org/wiki/FLOPS) of performance and 128 GB high [bandwidth](https://en.wikipedia.org/wiki/Bandwidth_(computing)) memory (HBM). Cloud TPU v3 Pods offer 100+ [petaflops](https://en.wikipedia.org/wiki/FLOPS) of performance and 32 TB HBM.

In February 2018, Google announced that they were making TPUs available in beta on the [Google Cloud Platform](https://en.wikipedia.org/wiki/Google_Cloud_Platform).

### Edge TPU

In July 2018, the Edge TPU was announced. Edge TPU is Google's purpose-built [ASIC](https://en.wikipedia.org/wiki/Application-specific_integrated_circuit) chip designed to run TensorFlow Lite machine learning (ML) models on small client computing devices such as smartphones known as [edge computing](https://en.wikipedia.org/wiki/Edge_computing).

### TensorFlow Lite

In May 2017, Google announced a software stack specifically for mobile development, TensorFlow Lite. In January 2019, the TensorFlow team released a developer preview of the mobile GPU inference engine with OpenGL ES 3.1 Compute Shaders on Android devices and Metal Compute Shaders on iOS devices. In May 2019, Google announced that their TensorFlow Lite Micro (also known as TensorFlow Lite for Microcontrollers) and [ARM's](https://en.wikipedia.org/wiki/Arm_Holdings) uTensor would be merging.

### TensorFlow 2.0

As TensorFlow's market share among research papers was declining to the advantage of [PyTorch](https://en.wikipedia.org/wiki/PyTorch),[[30]](https://en.wikipedia.org/wiki/TensorFlow#cite_note-:9-30) the TensorFlow Team announced a release of a new major version of the library in September 2019. TensorFlow 2.0 introduced many changes, the most significant being TensorFlow eager, which changed the automatic differentiation scheme from the static computational graph to the "Define-by-Run" scheme originally made popular by [Chainer](https://en.wikipedia.org/wiki/Chainer) and later [PyTorch](https://en.wikipedia.org/wiki/PyTorch). Other major changes included removal of old libraries, cross-compatibility between trained models on different versions of TensorFlow, and significant improvements to the performance on GPU.

TensorFlow is an end-to-end open-source platform for machine learning developed by Google with many enthusiastic open-source contributors. TensorFlow is scalable and flexible to run on data centers as well as mobile phones. It can run on single-machine as well as multiple-machine in a distributed setting. In this article, we will explore the secret behind the extreme flexibility and scalability of TensorFlow

Some terms need to be understood first to understand TensorFlow architecture. The terms are

TensorFlow Servable, servable Streams, TensorFlow Models, Loaders, Sources, Manager, and

Core. The term and their functionality in the architecture of TensorFlow are described below.

The first layer of TensorFlow consists of the device layer and the network layer.

The device layer contains the implementation to communicate to the various devices like GPU, CPU, TPU in the operating system where TensorFlow will run. Whereas the network layer has implementations to communicate with different machines using different networking protocols in the Distributable Trainable setting.

The second layer of TensorFlow contains kernel implementations for applications mostly used in machine learning.

The third layer of TensorFlow consists of distributed master and dataflow executors. Distributed Master has the ability to distribute workloads to different devices on the system. Whereas data flow executor performs the data flow graph optimally.

The next layer exposes all the functionalities in the form of API which is implemented in C language. C language is chosen because it is fast, reliable, and can run on any operating system.

The fifth layer provides support for Python and C++ clients.

And the last layer of TensorFlow contains training and inference libraries implemented in python and C++

**3.4 COMPUTER VISION TECHNIQUES:**

**FEATURE EXTRACTION:**

Extracting relevant features from hand images is crucial for understanding hand gestures. Features may include hand shape, size, position, orientation, and motion.

Techniques such as edge detection, corner detection, and blob analysis can be used to identify salient features of the hand.

**SEGMENTATION:**

Segmenting the hand region from the background or other objects in the scene is essential for isolating the hand gesture.

Thresholding, contour detection, and background subtraction are common techniques used for hand segmentation.

**GESTURE REPRESENTATION:**

Representing hand gestures in a suitable format for recognition is necessary. This representation may involve spatial, temporal, or spatiotemporal features.

Spatial representations may include hand shape descriptors, hand posture features, or hand landmarks detected using techniques like key point detection.

Temporal representations capture the dynamic aspects of gestures over time, such as motion trajectories or velocity profiles.

**3.4** **MACHINE LEARNING MODELS:**

Machine learning and deep learning techniques are often applied to learn models for hand gesture recognition.

Traditional machine learning algorithms, such as Support Vector Machines (SVMs), kNearest Neighbors (k-NN), or Random Forests, can be trained on handcrafted features extracted from hand images.

**SUPPORT VECTOR MACHINES(SVM):**

SVMs are popular for classification tasks, including hand gesture recognition.

SVMs find the hyperplane that best separates different classes of hand gestures in a highdimensional feature space.

Features extracted from hand images, such as Histogram of Oriented Gradients (HOG) or Scale-Invariant Feature Transform (SIFT), are often used as input to SVM classifiers.

**RANDOM FOREST:**

Random Forests are an ensemble learning method that combines multiple decision trees. Each decision tree is trained on a random subset of the training data, and the final prediction is made by aggregating the predictions of individual trees.

Random Forests can handle high-dimensional feature spaces and are robust to overfitting.

**NEURAL NETWORKS:**

Neural networks, including feedforward neural networks and multilayer perceptrons (MLPs), can be trained for hand gesture recognition.

These models consist of multiple layers of interconnected neurons, which learn hierarchical representations of hand images.

Various activation functions, such as ReLU, sigmoid, or tanh, can be used in neural networks.

Training neural networks requires labelled data and optimization algorithms such as stochastic gradient descent (SGD) or Adam.

**3.5 PROPOSED MODEL**

**Input Layer**:

Input images of fixed size representing hand gestures (e.g., 128x128 pixels). **Convolutional Layers:**

Begin with a series of convolutional layers to extract features from the input images. Each convolutional layer applies a set of learnable filters to the input image, resulting in feature maps that capture different aspects of the image.

Use ReLU (Rectified Linear Unit) activation function to introduce non-linearity. Incorporate pooling layers (e.g., max pooling) to down sample the feature maps and reduce the spatial dimensions while retaining the most important information. **Normalization Layer:**

Apply batch normalization to normalize the activations of the previous layer, which helps in stabilizing and accelerating the training process.

**Dropout Layer:**

Introduce dropout regularization to prevent overfitting by randomly dropping a fraction of the neurons during training. **Flattening Layer:**

Flatten the output of the last convolutional layer into a one-dimensional vector to prepare for the fully connected layers.

**Fully Connected Layers:**

Add one or more fully connected layers to process the flattened feature vector. Use ReLU activation for these layers.

**Output Layer:**

Include an output layer with softmax activation to produce probabilities for each class (i.e., each possible hand gesture).

**Model Compilation:**

Compile the model using appropriate loss function (e.g., categorical cross-entropy for multi-class classification) and optimizer (e.g., Adam optimizer). **Model Training:**

Train the model on a labelled dataset of hand gesture images, using techniques such as mini-batch gradient descent and backpropagation.

Monitor training performance and adjust hyperparameters as needed to improve accuracy.

**Evaluation:**

Evaluate the trained model on a separate validation dataset to assess its performance in terms of accuracy, precision, recall, and other relevant metrics. **Prediction**:

Once trained, the model can be used for predicting hand gestures in real-time or on new unseen data.

This proposed architecture can be adjusted and optimized based on the specific requirements of the hand gesture recognition task and the available dataset. Experimentation with different architectures, layer configurations, activation functions, and regularization techniques may be necessary to achieve optimal performance.

**3.6 BENEFITS OF THIS SYSTEM:**

Gesture recognition offers a path for computers to begin to better understand and interpret [human](https://en.wikipedia.org/wiki/Computer_processing_of_body_language) body languag[e,](https://en.wikipedia.org/wiki/Computer_processing_of_body_language) previously not possible through [text](https://en.wikipedia.org/wiki/Text_user_interface) or unenhanced [graphical](https://en.wikipedia.org/wiki/Graphical_user_interfaces) (GUI) user interfaces.

**FASTER INTERPRETATION:**

Natural flowing hand gestures help the audience interpret the message more effectively. [By activating multiple areas of the brain simultaneously, listeners both hear and see the message, enhancing comprehension.](https://sophiezadeh.com/body-language-blog/hand-gestures-in-communication)

**TOUCH LESS INTERFACES:**

Beyond presentations, hand-gesture applications have advantages in human-machine interaction. [They allow touchless interaction, which is especially useful in healthcare environments and for overcoming physical handicaps.](https://cacm.acm.org/research/vision-based-hand-gesture-applications/)

**ENHANCED HUMAN-COMPUTER INTERACTION (HCI):**

Hand gesture recognition provides a natural and intuitive means of interacting with computers, smartphones, and other electronic devices.

Users can control interfaces, navigate menus, and perform actions using simple hand gestures, reducing the need for physical input devices such as keyboards or mice. Gesture-based HCI can improve accessibility for individuals with physical disabilities or limitations, enabling them to interact with technology more effectively.

**VIRTUAL AND AUGMENTED REALITY (VR/AR):**

In VR and AR applications, hand gesture recognition enables users to interact with virtual environments and manipulate objects using hand movements.

Gesture-based interactions enhance immersion and realism in VR/AR experiences, making them more intuitive and engaging.

Hand gestures can be used for tasks such as selecting objects, grabbing and moving items, and gesturing to communicate with virtual characters or avatars.

**3.7 System Workflow**

The typical workflow of the system can be described as follows:

1.Data Collection: Gather a dataset of sign language gestures. This can include recording videos or capturing images of various hand gestures commonly used in sign language.

2.Data Preprocessing: Preprocess the collected data. Steps may include resizing images, normalizing pixel values, and augmenting the dataset through techniques like rotation, flipping, or adding noise to improve model generalization.

3.Hand Landmark Detection: Utilize the MediaPipe Hand Tracking module to detect and localize hand landmarks in each frame of the input video or image.

4. Feature Extraction: Extract relevant features from the detected hand landmarks. This may involve converting landmark coordinates into a suitable format for training the recognition model.

5. Model Selection: Choose an appropriate machine learning or deep learning model architecture for the task. CNNs are commonly used for image-based tasks, but you may also consider other models such as recurrent neural networks for sequence-based approaches.

6. Integration with OpenCV : Use OpenCV to capture live video from a camera feed or process pre-recorded video files. For each frame of the video, apply hand landmark detection using MediaPipe. Pass the detected landmarks through the trained model to classify the sign language gesture.

7. Result Output: Display the live video stream captured from the camera using OpenCV. This provides the user with visual feedback of their hand gestures.

**3.8 Implementation Details**

**3.8.1 Tools and Libraries**

1. OpenCV (cv2): OpenCV is a popular open-source computer vision library. It provides various functions for image and video processing, including capturing frames from cameras, image manipulation, object detection.
2. MediaPipe: offering pre-trained models and tools for real-time hand tracking.
3. Hand Tracking Module : The HandTrackingModule provides functionalities for hand detection and tracking in images and videos.

4.Numpy (np): NumPy is a fundamental package for numerical computing in Python.

3.8.**2 Programming Language**

- The system is implemented in Python due to its extensive support for the required libraries and ease of development.

**3.8.3 Development Environment**

- The system can be developed and executed in any standard Python development environment. It is compatible with Windows, macOS, and Linux operating systems.

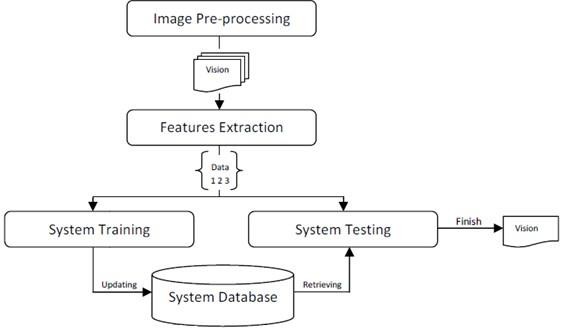
**3.9 Conclusion**

The project leverages OpenCV and the cvzone library's HandTrackingModule to capture and process hand images in real-time. With the aid of numpy and math for numerical and mathematical operations, it extracts hand regions from video frames, resizes them, and overlays them onto a white canvas. The system allows for the intuitive capture and storage of hand images upon the press of the 's' key, facilitating potential applications in gesture recognition and interactive interfaces.

**CHAPTER 4**

# ARCHITECTURE

## 4.1 ARCHITECTURE OF THE SYSTEM



**Fig:-4.1 Architecture of System**

**Working of Gesture Recognition:**

**Input Methods**: Cameras and motion sensors watch and record how you move. **Data Analysis**: Computers use special programs an[d machine learning](https://www.geeksforgeeks.org/ml-machine-learning/) to determine what these movements mean, tracking and interpreting them to match certain commands or actions.

**Applications**: This technology is everywhere, from video games where you control the action with your body, to virtual reality experiences, healthcare for guiding surgeries, or cars where a simple gesture can control the system.

**Technological Advancements**: Thanks to better AI and sensor tech, gesture recognition is becoming faster and more accurate, making gadgets smarter and easier to use. **User Experience**: This tech makes using devices more natural and fun, as you can control them just by moving, without touching anything.

**4.2 Arcitecture of Math Module:**

Math Module consists of mathematical functions and constants. It is a built-in module made for mathematical tasks.

The math module provides the math functions to deal with basic operations such as addition(+), subtraction(-), multiplication(\*), division(/), and advanced operations like trigonometric, logarithmic, and exponential functions.

CONSTANTS IN MATH MODULE:

The Python math module provides various values of various constants like pi, and tau. We can easily write their values with these constants. The constants provided by the math module are :

Euler’s Number

Pi

Tau

Infinity

Not a Number (NaN)

: NUMERICAL FUNCTIONS :

In this section, we will deal with the functions that are used with number theory as well as representation theory such as finding the factorial of a number. We will discuss these numerical functions along with examples and use-cases.

1. Finding the ceiling and the floor value

Ceil value means the smallest integral value greater than the number and the floor value means the greatest integral value smaller than the number. This can be easily calculated using the ceil() and floor() method respectively.

2. Finding the factorial of the number

Using the factorial() function we can find the factorial of a number in a single line of the code. An error message is displayed if number is not integral.

### 4.3 Arciteccture of MediaPipe

MediaPipe is an open-source framework for building pipelines to perform computer vision inference over arbitrary sensory data such as video or audio. Using MediaPipe, such a perception pipeline can be built as a graph of modular components. MediaPipe is currently in alpha at v0.7, and there may still be breaking API changes. Stable APIs are expected by v1.0

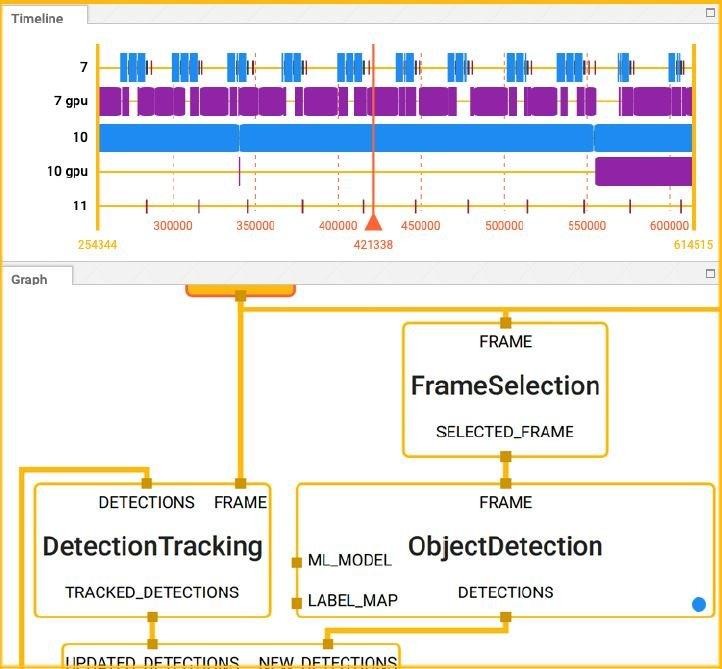
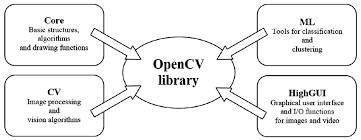


Fig:-4.2 MediaPipe Architecture

The MediaPipe framework is mainly used for rapid prototyping of perception pipelines with AI models for inferencing and other reusable components. It also facilitates the deployment of computer vision applications into demos and applications on different hardware platforms. The configuration language and evaluation tools enable teams to incrementally improve computer vision pipelines

## 4.4 OPEN CV

OpenCV, short for Open Source Computer Vision Library, is an essential toolkit for anyone working with computer vision and machine learning. It's open-source, which means anyone can use and tweak it, fitting for all sorts of projects, from big companies like Google to smaller startups and academic research



**Fig:-4.3 Open CV Architecture**

**Motion Understanding and Object Detection:**

OpenCV's algorithms for motion understanding and object detection can be used in surveillance, autonomous vehicles, and robotics. For example, they could be used to develop a motion detection system that monitors security camera feeds. Such a system could alert operators to suspicious movements in real time. In the image below, a technique called background subtraction detects motion and thus knows that an object (in this case, a dog) has entered the image.

The development of OpenCV Library started in 1999 at Intel Research Labs. It was first created in C and C++ languages and was later expanded to include support for modern programming languages like Python and many others. Created by Gary Bradski, the library mainly aims to provide a common infrastructure for various computer vision applications and accelerate research in the field.

### OpenCV in the 2000s

The first version was publicly released in 2000, version 1.0, and it offered many image processing and analysis functionalities. Six years later, version 1.1 introduced the new C++ interface, which made it more accessible to developers and enabled faster prototyping. In 2008, version 1.5 made use of the power of graphical processing units to accelerate computationally intensive tasks. This was done by adding support for GPU acceleration.

#### The Evolution in the 2010s

OpenCV released version 2.0 in 2010, and it surpassed everything that came before it. It gave developers enhanced performance, a modular structure, and compatibility with various platforms like Windows, Linux, Mac, and mobile devices. OpenCV brought major changes in 2015 with version 3.0. It had a new C++11 interface, improved Python bindings, and also integrated the latest machine-learning algorithms at the time. Later on, in 2018, deep learning capabilities and a DNN module for efficient neural network inference were introduced with Version 4.0.

#### The Present (2020s)

OpenCV is continually improving. At the time of writing this article, OpenCV recently introduced enhancements to its object detection modules along with a new object tracking API called

TrackerVit, which is based on vision transformers. Their latest version, v4.9, also had enhanced Android support capabilities, experimental CUDA language support, and the latest AppleVisionOS platform support.

**4.5 PYTHON TIME MODULE :**

As the name suggests Python time module allows to work with time in Python. It allows functionality like getting the current time, pausing the Program from executing, etc. So before starting with this module, we need to import it.

Importing time module

The time module comes with Python’s standard utility module, so there is no need to install it externally. We can simply import it using the import statement.

import time

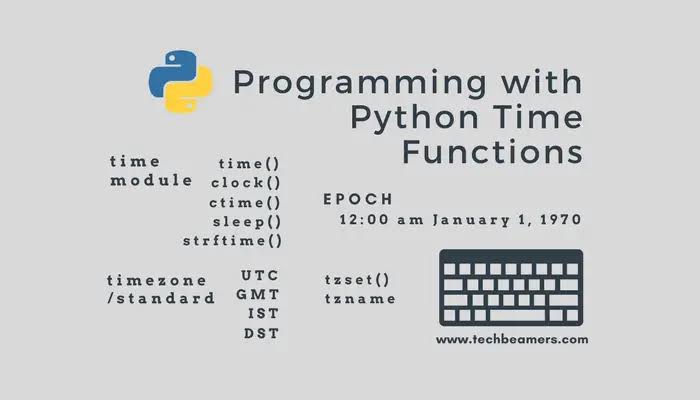
What is epoch?

The epoch is the point where the time starts and is platform-dependent. On Windows and most Unix systems, the epoch is January 1, 1970, 00:00:00 (UTC), and leap seconds are not counted towards the time in seconds since the epoch. To check what the epoch is on a given platform we can use time.gmtime(0).

Getting epoch

The code uses the time module to print the result of time.gmtime(0), which represents the time in the GMT (Greenwich Mean Time) timezone at the Unix epoch .

The code uses the time module to retrieve the current time in seconds since the Unix epoch .



**Fig:-4.4 Time Module**

**4.6 HAND DETECTOR MODULE**

**Hand Detection and Tracking with MediaPipe in Python**

**MediaPipe Overview**:

**MediaPipe** is an open-source, cross-platform machine learning framework developed by Google. It offers pre-trained ML solutions for various tasks, including **hand detection**.

The **Hand module** in MediaPipe detects and tracks hands in images or video frames. It returns a total of **21 key points** (landmarks) for each detected hand.These key points represent specific anatomical points on the hand, such as the wrist, thumb, index finger, middle finger, ring finger, and pinky finger.

1. **Key Points (Landmarks)**:
   * **Wrist**: The base of the hand.
   * **Thumb (Points 2–5)**: Representing the tip, knuckle, and two points along the thumb.
   * **Index Finger (Points 6–9)**: Representing the tip, knuckle, and two points along the index finger.
   * **Middle Finger (Points 10–13)**: Representing the tip, knuckle, and two points along the middle finger.
   * **Ring Finger (Points 14–17)**: Representing the tip, knuckle, and two points along the ring finger.
   * **Pinky Finger (Points 18–21)**: Representing the tip, knuckle, and two points along the pinky finger.
2. **Using MediaPipe for Hand Detection**:
   * Install the necessary packages:
     + OpenCV: pip install opencv-python
     + MediaPipe: pip install mediapipe
3. **Application**:

By analyzing the relative positions and movements of these key points over time, you can infer gestures or hand poses.Use cases include gesture recognition, hand tracking, virtual reality interactions, and more.

**CHAPTER 5**

**SYSTEM SPECIFICATIONS**

**5.1 Hardware Requirements**

To ensure optimal performance and functionality, the system requires the following hardware specifications:

**5.1.1 Minimum Hardware Requirements**

* Processor: Dual-core processor (2.0 GHz or higher)
* RAM: 8 GB
* Storage: 10 GB free disk space
* Image Input/Output: Camera and video
* Operating System: Windows 11 or later, macOS 10.12

**5.2 Software Requirements**

The system relies on several software components to function effectively. The following software specifications are required:

**5.2.1 Operating System**

**Compatibility:** Windows, macOS, and Linux

**5.2.2 Programming Language**

**Python**: Version 3.6 or higher

**5.2.3 Libraries and Dependencies**

### OPEN CV

OpenCV is a Python library which is intended to tackle PC vision issues. It supports various languages and has been the most important library to deal with recognition type projects where frame or video needs To captured an dealt with whether it Is object detection or motion detection.

OpenCV is a python library which is very helpful in calculating hand tracing and it is equipped with certain functions which are the steps towards creating a model having signal identification and those functions include video capturing, background removal, image editing like (resizing, rotation ), plotting motion detection graph etc.

#### NumPy

NumPy is a Python module. The name NumPy represents :Numerical

Python and it is utilized. It is an expansion module for Python, generally written in C. This guarantees the precompiled logical and numerical limits and functionalities of Numpy guarantee remarkable execution speed.

Numpy is mostly used for performing calculations using certain functions it provides like multiply, divide, power etc. It is basically used for performing complex calculations with ease like dot product, summation etc. It is a very efficient module and makes the work easier.



Fig:-5 NumPy Description.

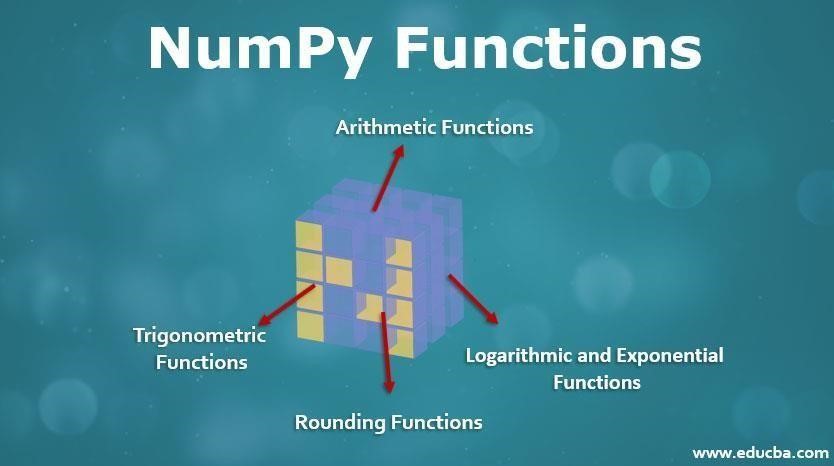
e

Fig:-5.1 NumPy Functions.

**OPENCV**

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being an Apache 2 licensed product, OpenCV makes it easy for businesses to utilize and modify the code.

**MEDIAPIPE**

Developed by Google, it offers a range of solutions including hand tracking Real-time hand and finger tracking capabilities.Easy to integrate with other machine learning models**.**

**5.3 Conclusion**

The system specifications outlined in this chapter provide a comprehensive overview of the hardware and software requirements necessary for the successful implementation and operation of the project. By adhering to these specifications, the system can be configured to perform optimally, ensuring a smooth and efficient user experience.

**CHAPTER 6**

**SYSTEM TESTING**

**6.1 Introduction**

System testing is a critical phase in the software development lifecycle. It involves validating that the system meets all specified requirements and functions correctly in all anticipated scenarios. This chapter outlines the testing strategy, types of testing performed, test cases, and results for the project.

**6.2 Testing Strategy**

The testing strategy for the project includes the following stages:

**1. Unit Testing**: Unit testing focuses on verifying individual components or functions of the system to ensure they work correctly in isolation.

**2. Integration Testing**: Testing the integration of different modules to ensure they work together as expected.

**3. System Testing**: Testing the complete system to verify it meets all requirements.

**4. User Acceptance Testing (UAT):** Testing the system with actual users to ensure it meets their needs and expectations.

**6.3 Types of Testing**

**6.3.1 Unit Testing**

Unit testing involves testing individual functions and methods to ensure they work as expected. Each module in the system is tested independently.

**6.3.2 Integration Testing**

Integration testing involves testing the interactions between different modules. This ensures that data flows correctly between modules and that they work together seamlessly.

**6.3.3 System Testing**

System testing involves testing the complete, integrated system to verify it meets all specified requirements. This includes functional and non-functional testing.

**6.3.4 User Acceptance Testing (UAT)**

UAT involves testing the system with actual users to ensure it meets their needs and expectations. Feedback from users is used to make final adjustments before deployment.

**6.5 Test Results**

The testing process yielded the following results:

**Unit Testing:** All unit tests passed, indicating that individual components function correctly.

**Integration Testing:** All integration tests passed, confirming that modules work together seamlessly.

**System Testing**: All system tests passed, verifying that the system meets all specified requirements.

**User Acceptance Testing:** Users provided positive feedback, confirming that the system meets their needs and is user-friendly.

**6.6 Conclusion:**

The system testing phase was successful, with all test cases passing and users providing positive feedback. The thorough testing strategy ensured that the system functions correctly, meets all requirements, and provides a satisfactory user experience. The system is now ready for deployment and use in real-world scenarios.

**CHAPTER 7**

**SYSTEM IMPLEMENTATION**

**7.1 Introduction**

System implementation is the process of deploying and configuring the software in a production environment and ensuring that it is fully operational. This chapter covers the steps involved in the implementation phase, including setup, installation, and deployment procedures, as well as training and user support.

**7.2 Implementation Plan**

The implementation plan outlines the steps necessary to deploy the system in a live environment. The plan includes the following key stages:

1**.** Preparation and Setup

2. Installation of Software

3. Configuration

4. Data Migration

5. Testing in Production

6. User Training and Support

**7.3 Preparation and Setup**

**7.3.1 Hardware Preparation**

* Ensure that all required hardware components are available and functional.
* Verify that the hardware meets the minimum and recommended specifications outlined in Chapter 6.

**7.3.2 Software Preparation**

Confirm that all required software and libraries are available and compatible with the system. Prepare installation scripts and documentation for the deployment process.

**7.4 Installation of Software**

**7.4.1 Installing Python and Libraries**

**1. Install Python:**

Download and install Python 3.6 or higher from the official Python website.

Verify the installation by running `python --version` in the command line.

**2. Set Up Virtual Environment:**

Create a virtual environment to manage project dependencies

python -m venv env

Activate the virtual environment:

Windows: `.\env\Scripts\activate`

macOS/Linux: `source env/bin/activate`

**3. Install Required Libraries:**

- Install the necessary libraries using pip:

pip install numpy scipy

**7.5 Configuration**

**7.5.1 System Configuration**

* Configure the system settings according to the requirements specified in Chapter 6.
* Ensure that the microphone and speakers are properly configured and tested for audio input and output.

**7.5.2 Application Configuration**

Configure the application settings, such as summarization parameters, voice settings for speech synthesis, and directory paths for PDF files and output summaries.

**7.6 Data Migration**

If the system needs to handle existing data, follow these steps for data migration:

**1. Backup Existing Data:**

Create backups of any existing data to ensure it is not lost during the migration process.

**2. Transfer Data:**

Transfer existing files and other relevant data to the new system.

**3. Verify Data Integrity:**

Verify that the transferred data is intact and accessible in the new system.

**7.7 Testing in Production**

Before going live, perform the following tests in the production environment:

**1. Functional Testing:**

Test all functionalities of the system to ensure they work as expected in the production environment.

**2. Performance Testing:**

Test the system's performance to ensure it can handle the expected load without issues.

**3. User Acceptance Testing:**

Conduct final user acceptance testing to verify that the system meets user requirements and expectations.

**7.8 User Training and Support**

**7.8.1 User Training**

* Provide training sessions for users to familiarize them with the system's features and functionalities.
* Create user manuals and documentation to assist users in operating the system.

**7.8.2 User Support**

* Establish a support system to assist users with any issues or questions they may have.
* Provide contact information for technical support and troubleshooting.

**7.9 Deployment**

After successful testing and user training, proceed with the full deployment of the system:

**1. Go Live:**

Deploy the system to the production environment and make it accessible to users.

**2. Monitor System:**

* Monitor the system closely for any issues or bugs that may arise after deployment.
* Address any issues promptly to ensure smooth operation.

**3. Collect Feedback:**

* Collect feedback from users to identify areas for improvement and gather suggestions for future enhancements.

**7.10 Conclusion**

The system implementation phase involves a comprehensive plan to deploy and configure the software in a production environment. By following the steps outlined in this chapter, the system can be successfully implemented, ensuring that it is fully operational and meets user requirements. Effective training and support will help users make the most of the system's features, leading to a smooth transition and positive user experience.

**CHAPTER 8**

# RESULTS

Various machine learning models are used for sign detection. These models are evaluated based on parameters like accuracy, recall, F1 score, etc. Among the utilized models, it is observed that SVM outperformed other machine learning techniques such as Naive Bayes, KNN, Decision Tree, etc. by achieving an accuracy of 98.65% (training) and 98.35% (testing) as shown in table 0. The reason it outperformed is because of its effectiveness in high- dimensional spaces where it draws a hyperplane boundary in order to classify the labels. It is also computationally less extensive and works well for image analysis tasks.

The below table shows the values of training and testing accuracy along with Recall, F1Score and Precision for different tried models:

Table 0 Results for various ML algorithms

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Model** | **Train (%)** | **Test (%)** | **Recall** | **F1 Score** | **Precision** |
| SVM | **99.70** | **98.975** | 0.98 | 0.98 | 0.98 |
| Random Forest | 99.89 | 97.50 | 0.97 | 0.97 | 0.97 |
| Decision Tree | 99.89 | 91.52 | 0.91 | 0.91 | 0.91 |
| Naïve Bayes | 50.63 | 50.84 | 0.50 | 0.50 | 0.50 |
| KNN | 97.62 | 96.39 | 0.96 | 0.96 | 0.96 |

Fig Table:-8 Accuracy of outputs

# CONCLUSION AND FUTURE ENHANCEMENTS

## CONCLUSION

Individuals with hearing disabilities often face significant challenges in communicating with people who can hear. One of the most effective ways for them to communicate is through sign language. However, for people who do not know sign language, understanding what is being communicated can be a significant challenge. This communication gap can have a detrimental impact on the social and emotional well-being of individuals with hearing disabilities, making it difficult for them to engage fully in society.

The proposed Sign Language Recognition system offers an innovative solution to the communication gap between individuals with hearing disabilities and those who can hear.

The proposed system successfully recognizes sign language with high accuracy, with an

SVM model achieving a classification accuracy of 98.975%. Moreover, the use of Google's MediaPipe palm detector method has made the system accessible to people without any special hardware, which is a significant advantage.

The proposed method's potential for practical applications is considerable, and it has the capacity to improve the quality of life for individuals with hearing disabilities, helping to bridge the communication gap between them and the rest of the world. Future work will expand the current system to add more indicators and create a complete and reliable system for mobile platforms. Additionally, the proposed method can be adapted for use in other Indian regional languages, such as Hindi, Marathi, Sindhi, Telugu, and more. Although there are still some research gaps that need to be addressed, such as improving the system's accuracy in recognizing signs for complex phrases and developing a portable and affordable device for practical use in daily life, the proposed Sign Language Recognition system offers a promising step towards creating a more inclusive society. With further development and refinement, this system can play a significant role in breaking down communication barriers and facilitating greater accessibility and understanding for individuals with hearing disabilities.

**FUTURE ENHANCEMENTS**

While current hand gesture recognition systems are quite advanced, there are several areas for future enhancements to improve their accuracy, efficiency, and applicability: 1. Improved Accuracy and Robustness Advanced Algorithms: Incorporate more sophisticated machine learning and deep learning algorithms to enhance gesture recognition accuracy, especially in complex and dynamic environments. Data Augmentation: Use extensive data augmentation techniques to train models on a wider variety of gestures, lighting conditions, and backgrounds to improve robustness. 2. Real-time Performance Optimization: Optimize algorithms and models for real-time performance, reducing latency and ensuring smooth interaction. Edge Computing: Implement edge computing solutions to process gestures locally on devices, minimizing reliance on cloud-based processing and reducing response times. 3. User Adaptation and Personalization 33 Personalized Models: Develop models that can adapt to individual users' gesture patterns, improving recognition accuracy for each user. Feedback Mechanisms: Implement feedback mechanisms that allow users to correct misrecognized gestures, helping the system learn and improve over time. 4. Multimodal Interaction Integration with Other Modalities: Combine gesture recognition with other interaction modalities such as voice recognition, facial expressions, and eye tracking for more comprehensive and natural user interfaces. Context Awareness: Enhance systems to be context-aware, understanding the environment and 10.3 Conclusion The completion of this project demonstrates the feasibility and effectiveness of integrating various technologies to create a comprehensive processing system. The project has successfully met its objectives, providing users with an innovative tool for image extraction, summarization, and image generation from images and video. By addressing the outlined future enhancements, the system can evolve to become even more powerful, accessible, and user-friendly, catering to a broader range of needs and preferences.

**REFERENCES**

In compiling the research and development of this project, several resources, tools, and libraries were utilized. The following references provide a comprehensive list of all the materials that were consulted:

**References**

# 1.Real-time Hand Gesture Recognition using TensorFlow & OpenCV

# <https://techvidvan.com/tutorials/hand-gesture-recognition-tensorflow-opencv/>

# 2. Gesture based Real-Time Sign Language Recognition System

**Published in:**[2022 International Conference on Connected Systems & Intelligence (CSI)](https://ieeexplore.ieee.org/xpl/conhome/9923920/proceeding)

**Date of Conference:**31 August 2022 - 02 September 2022

**Date Added to IEEE *Xplore*:**01 November 2022

**ISBN Information:**

**DOI:**[10.1109/CSI54720.2022.9924024](https://doi.org/10.1109/CSI54720.2022.9924024)

**Publisher:**IEEE

**Conference Location:**Trivandrum, India

# 3. Hand Gesture based Sign Language recognition using Mediapipe

**Published in:**[2022 International Conference on Artificial Intelligence, Big Data, Computing and Data Communication Systems (icABCD)](https://ieeexplore.ieee.org/xpl/conhome/9855894/proceeding)

**Date of Conference:**04-05 August 2022

**Date Added to IEEE *Xplore*:**22 August 2022

**Authors** :[Tshephisho Joseph Sefara](https://ieeexplore.ieee.org/author/37086217212); [Mahlatse Mbooi](https://ieeexplore.ieee.org/author/37089497176); [Katlego Mashile](https://ieeexplore.ieee.org/author/37089496237); [Thompho Rambuda](https://ieeexplore.ieee.org/author/37089496069); [Mapitsi Rangata](https://ieeexplore.ieee.org/author/37089499252).

Authors :[Pooja Raundale](https://ieeexplore.ieee.org/author/37085829039); [Himanshu Shekhar](https://ieeexplore.ieee.org/author/37089039877)

Google Referenceses :--

1. [Hand Detection Tracking in Python using OpenCV and MediaPipe](https://medium.com/@gautamaditee/hand-recognition-using-opencv-a7b109941c88)
2. [Google AI Edge - Hand Landmarker](https://ai.google.dev/edge/mediapipe/solutions/vision/hand_landmarker/python)
3. [HandDetectorModule on PyPI](https://pypi.org/project/HandDetectorModule/)
4. [Google AI Edge - Gesture Recognizer](https://ai.google.dev/edge/mediapipe/solutions/vision/gesture_recognizer/python)
5. [Hand Recognition with Python: Guide with Code Samples](https://medium.com/@lota.pipeline/hand-recognition-with-python-guide-with-code-samples-a0b17f4cd813)

https://www.google.com/amp/s/www.geeksforgeeks.org/python-math-module/amp/

https://www.geeksforgeeks.org/python-time-module/

https://www.google.com/search?q=hand+detector+module+in+python&oq=hand+detector+module+in+python+&gs\_lcrp=EgZjaHJvbWUyBggAEEUYOTIJCAEQABgNGIAEMg0IAhAAGIYDGIAEGIoFMg0IAxAAGIYDGIAEGIoFMg0IBBAAGIYDGIAEGIoFMg0IBRAAGIYDGIAEGIoFMgoIBhAAGIAEGKIEMgoIBxAAGIAEGKIEMgoICBAAGIAEGKIEMgoICRAAGIAEGKIE0gEIODk4NmowajmoAg6wAgE&client=ms-android-xiaomi-rev1&sourceid=chrome-mobile&ie=UTF-8

**APPENDIX**

**A. SOURCE CODE**

**DATACOLLECTION.PY**

import cv2

from cvzone.HandTrackingModule import HandDetector

import numpy as np

import math

import time

cap = cv2.VideoCapture(0)

detector = HandDetector(maxHands=1)

offset = 20

imgSize = 300

counter = 0

folder = "data/fun"

while True:

    success, img = cap.read()

    hands, img = detector.findHands(img)

    if hands:

        hand = hands[0]

        x, y, w, h = hand['bbox']

        imgWhite = np.ones((imgSize, imgSize, 3), np.uint8)\*255

        imgCrop = img[y-offset:y + h + offset, x-offset:x + w + offset]

        imgCropShape = imgCrop.shape

        aspectRatio = h / w

        if aspectRatio > 1:

            k = imgSize / h

            wCal = math.ceil(k \* w)

            imgResize = cv2.resize(imgCrop, (wCal, imgSize))

            imgResizeShape = imgResize.shape

            wGap = math.ceil((imgSize-wCal)/2)

            imgWhite[:, wGap: wCal + wGap] = imgResize

        else:

            k = imgSize / w

            hCal = math.ceil(k \* h)

            imgResize = cv2.resize(imgCrop, (imgSize, hCal))

            imgResizeShape = imgResize.shape

            hGap = math.ceil((imgSize - hCal) / 2)

            imgWhite[hGap: hCal + hGap, :] = imgResize

        cv2.imshow('ImageCrop', imgCrop)

        cv2.imshow('ImageWhite', imgWhite)

    cv2.imshow('Image', img)

    key = cv2.waitKey(1)

    if key == ord("s"):

        counter += 1

        cv2.imwrite(f'{folder}/Image\_{time.time()}.jpg', imgWhite)

        print(counter)

**TEST.PY**

import cv2

from cvzone.HandTrackingModule import HandDetector

from cvzone.ClassificationModule import Classifier

import numpy as np

import math

cap = cv2.VideoCapture(0)

detector = HandDetector(maxHands=1)

classifier = Classifier("C:\practise codes\HandData\Model\keras\_model.h5" , "C:\practise codes\HandData\Model\labels.txt")

offset = 20

imgSize = 300

counter = 0

labels = ["Hello","I love you","No","Okay","Please","Thank you","Yes"]

while True:

    success, img = cap.read()

    imgOutput = img.copy()

    hands, img = detector.findHands(img)

    if hands:

        hand = hands[0]

        x, y, w, h = hand['bbox']

        imgWhite = np.ones((imgSize, imgSize, 3), np.uint8)\*255

        imgCrop = img[y-offset:y + h + offset, x-offset:x + w + offset]

        imgCropShape = imgCrop.shape

        aspectRatio = h / w

        if aspectRatio > 1:

            k = imgSize / h

            wCal = math.ceil(k \* w)

            imgResize = cv2.resize(imgCrop, (wCal, imgSize))

            imgResizeShape = imgResize.shape

            wGap = math.ceil((imgSize-wCal)/2)

            imgWhite[:, wGap: wCal + wGap] = imgResize

            prediction , index = classifier.getPrediction(imgWhite, draw= False)

            print(prediction, index)

        else:

            k = imgSize / w

            hCal = math.ceil(k \* h)

            imgResize = cv2.resize(imgCrop, (imgSize, hCal))

            imgResizeShape = imgResize.shape

            hGap = math.ceil((imgSize - hCal) / 2)

            imgWhite[hGap: hCal + hGap, :] = imgResize

            prediction , index = classifier.getPrediction(imgWhite, draw= False)

        cv2.rectangle(imgOutput,(x-offset,y-offset-70),(x -offset+400, y - offset+60-50),(0,255,0),cv2.FILLED)

        cv2.putText(imgOutput,labels[index],(x,y-30),cv2.FONT\_HERSHEY\_COMPLEX,2,(0,0,0),2)

        cv2.rectangle(imgOutput,(x-offset,y-offset),(x + w + offset, y+h + offset),(0,255,0),4)

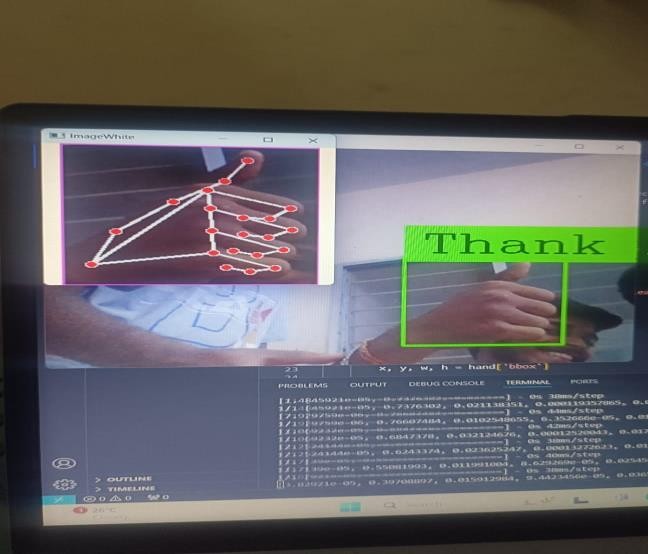
        cv2.imshow('ImageCrop', imgCrop)

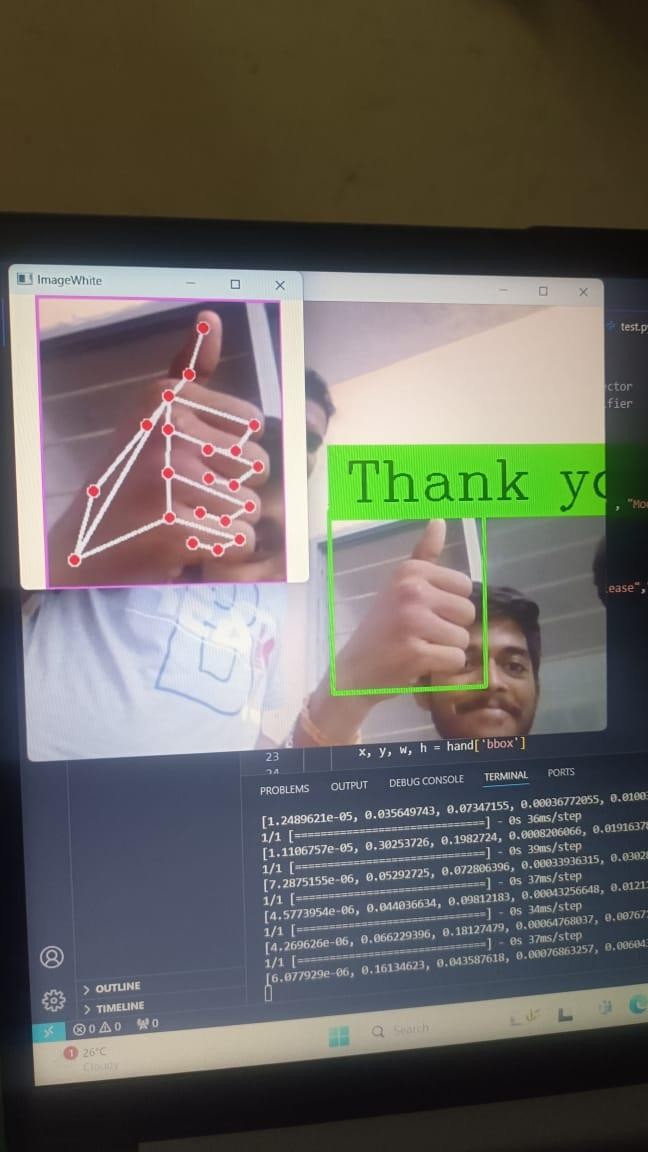
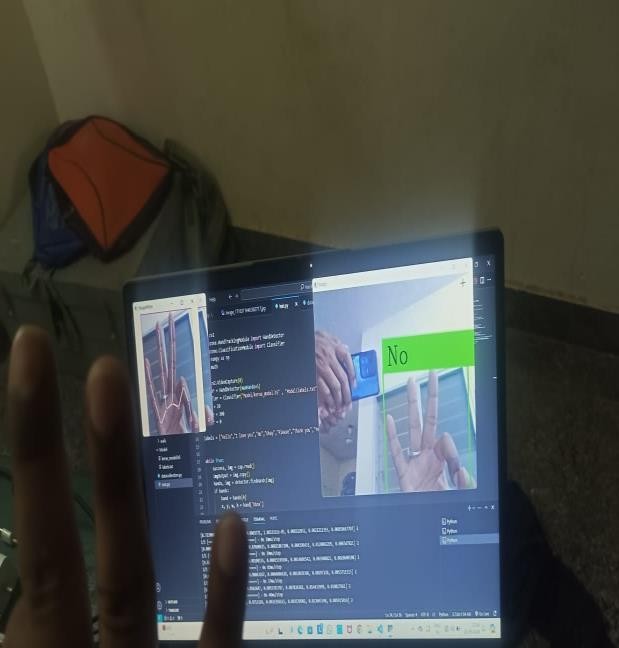
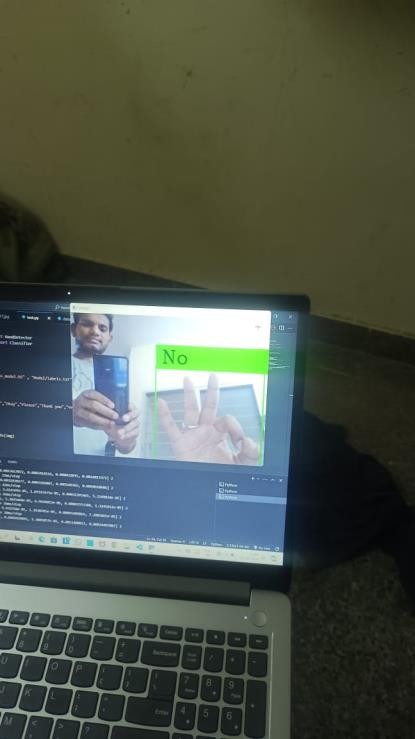
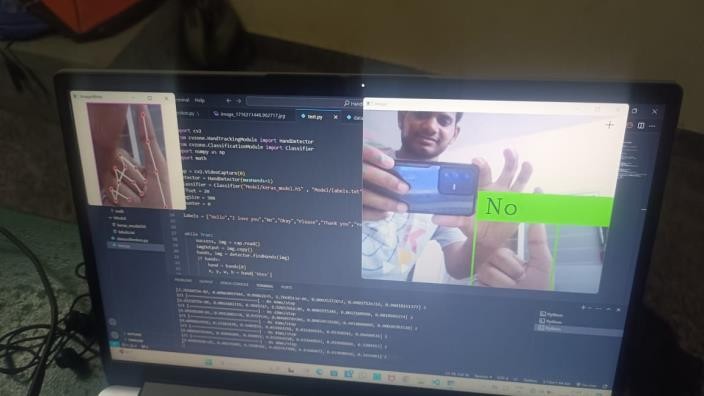
        cv2.imshow('ImageWhite', imgWhite)

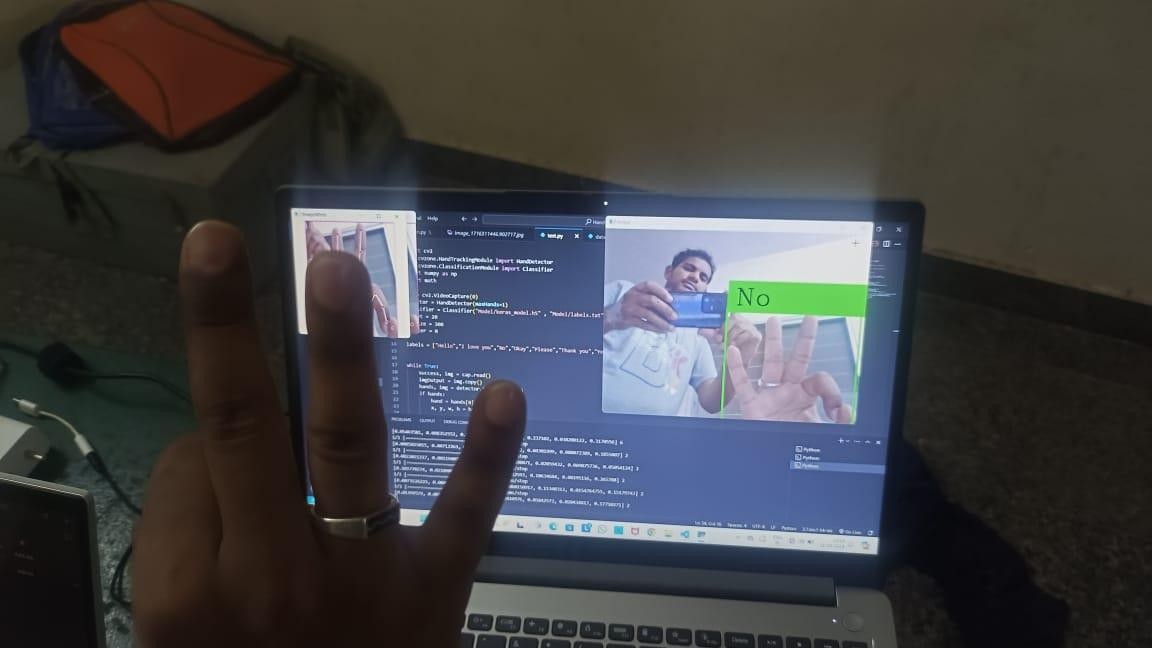
    cv2.imshow('Image', imgOutput)

    cv2.waitKey(1)

# B. SCREENSHOTS







**Title of the Project**: **HAND GESTURE BASED** **RECOGNITION SYSTEM**

**Name of the students**: S NISAR

Name of the Guide & Designation: **Dr.N.GAYATHRI DEVI.,M.Tech.,P.hd, / Associate Professor**

**TABLE 1: OUTCOME ATTAINED AND ITS JUSTIFICATION**

|  |  |
| --- | --- |
| **PO** | **Justification** |
| PO1 | The knowledge About the Hand Gesture Bsed Recognition System is known through this project work |
| PO2 | Analyzed the problems of accuracy through the algorithms |
| PO3 | Hand gesture based recognition How its going to work |
| PO4 | We used datasets and research-based data to provide valid conclusions |
| PO5 | We implemented our work with well appropriate techniques, good resources modern AI engineering tools the project. |
| PO6 | This solution increases the accuracy of the modules produced and Productivity for sustainable development of the society. |
| PO7 | This solution increases the accuracy of the algothims produced. Hence technological methods of resources happens. |
| PO8 | We followed the ethical principles. |
| PO9 | We worked in this project function effectively as a member of the project team. |
| PO10 | Oral and written communication skills are improved while planning, implementing and executing the entire project and till submission of the report. |
| PO11 | We demonstrated our knowledge and understanding of cost and time analysis required for carrying out the project. |
| PO12 | Facilitated ourselves in Lifelong learning to improve technical knowledge and competence in the chosen area of the project. |

**Evaluation Rubrics for Project work:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Rubric (CO)** | **Excellent (Wt = 3)** | **Good (Wt = 2)** | **Fair (Wt = 1)** |
| ***Selection of Topic (CO1)*** | Select a latest topic through complete knowledge of facts and concepts. | Select a topic through partial knowledge of facts and concepts. | Select a topic through improper knowledge of facts and concepts. |
| ***Analysis and Synthesis (CO2)*** | Thorough comprehension through analysis/ synthesis. | Reasonable comprehension through analysis/ synthesis. | Improper comprehension through analysis/ synthesis. |
| ***Problem Solving (CO3)*** | Thorough comprehension about what is proposed in the literature papers. | Reasonable comprehension about what is proposed in the literature papers. | Improper comprehension about what is proposed in the literature. |
| ***Literature Survey (CO4)*** | Extensive literature survey with standard references. | Considerable literature survey with standard references. | Incomplete literature survey with substandard references. |
| ***Usage of Techniques & Tools (CO5)*** | Clearly identified and has complete knowledge of techniques & tools used in the project work. | Identified and has sufficient knowledge of techniques & tools used in the project work. | Identified and has inadequate knowledge of techniques & tools used in project work. |
| ***Project work impact on Society (CO6)*** | Conclusion of project work has strong impact on society. | Conclusion of project work has considerable impact on society. | Conclusion of project work has feeble impact on society. |
| ***Project work impact on Environment (CO7)*** | Conclusion of project work has strong impact on Environment. | Conclusion of project work has considerable impact on environment. | Conclusion of project work has feeble impact on environment. |
| ***Ethical attitude (CO8)*** | Clearly understands ethical and social practices. | Moderate understanding of ethical and social practices. | Insufficient understanding of ethical and social practices. |
| ***Independent Learning (CO9)*** | Did literature survey and selected topic with a little guidance | Did literature survey and selected topic with considerable guidance | Selected a topic as suggested by the supervisor |
| ***Oral Presentation (CO10)*** | Presentation in logical sequence with key points, clear conclusion and excellent language | Presentation with key points, conclusion and good language | Presentation with insufficient key points and improper conclusion |
| ***Report Writing (CO10)*** | Status report with clear and logical sequence of chapters using excellent language | Status report with logical sequence of chapters using understandable language | Status report not properly organized |
| ***Time and Cost Analysis (CO11)*** | Comprehensive time and cost analysis | Moderate time and cost analysis | Reasonable time and cost analysis |
| ***Continuous learning (CO12)*** | Highly enthusiastic towards continuous learning | Interested in continuous learning | Inadequate interest in continuous learning |