REAL TIME LANGUAGE ASSISTANT FOR SIGN TRANSLATION

A project report submitted in partial fulfillment of the requirement for the Award of the Degree of

BACHELOR OF ENGINEERING

in

ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

By

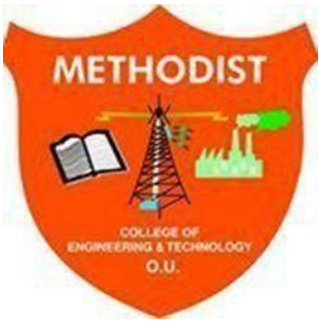
S. RANJITH KUMAR (160720747306)

P. NIKHITHA (160720747039)

P. LIKITHA (160720747302)

Under the Guidance of

Mrs . J SOWMYA,
Assistant Professor, Dept. of CSE



Department of Artificial Intelligence and Data Science
Methodist College of Engineering and Technology,
King Koti, Abids, Hyderabad-500001.
2023-2024

REAL TIME LANGUAGE ASSISTANT FOR SIGN TRANSLATION

A project report submitted in partial fulfillment of the requirement for the Award of the Degree of

BACHELOR OF ENGINEERING

in

ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

By

S. RANJITH KUMAR (160720747306)

P. NIKHITHA (160720747039)

P. LIKITHA (160720747302)

Under the Guidance of

Mrs . J SOWMYA,
Assistant Professor, Dept. of CSE

Department of Artificial Intelligence and Data Science
Methodist College of Engineering and Technology,
King Koti, Abids, Hyderabad-500001.

2023-2024

Methodist College of Engineering and Technology,
King Koti, Abids, Hyderabad-500001,

Department of Artificial Intelligence and Data Science

DECLARATION BY THE CANDIDATES

We, **S. RANJITH KUMAR (160720747306), P. NIKHITHA (160720747039) and P. LIKITHA (160720747302)** students of Methodist College of Engineering and Technology, pursuing Bachelor's degree in Artificial Intelligence and Data Science, hereby declare that this Project report entitled "**REAL TIME LANGUAGE ASSISTANT FOR SIGN TRANSLATION",** carried out under the guidance of **Mrs. J Sowmya** submitted in partial fulfillment of the requirements for the degree of Bachelor of Engineering in Artificial Intelligence and Data Science. This is a record work carried out by me and the results embodied in this report have not been reproduced/copied from any source.

S. RANJITH KUMAR (160720747306)

P. NIKHITHA (160720747039)

P. LIKITHA (160720747302)

Methodist College of Engineering and Technology,
King Koti, Abids, Hyderabad-500001.

Department of Artificial Intelligence and Data Science

CERTIFICATE BY THE SUPERVISOR

This is to certify that this Dissertation on project work entitled "**REAL TIME LANGUAGE ASSISTANT FOR SIGN TRANSLATION**" *by* **S. RANJITH KUMAR (160720747306), P. NIKITHA (160720747039) and P. LIKITHA (160720747302)** submitted in partial fulfillment of the requirements for the degree of Bachelor of Engineering in Artificial Intelligence and Data Science, during the academic year 2023-2024, is a bonafide record of work carried out by them.

Mrs.J . Sowmya,
Assistant Professor,
Dept. of CSE.

Date:

Methodist College of Engineering andTechnology,
King Koti, Abids, Hyderabad-500001.

Department of Artificial Intelligence and Data Science

CERTIFICATE BY HEAD OF THE DEPARTMENT

This is to certify that this Dissertation on project work entitled "**REAL TIME LANGUAGE ASSISTANT FOR SIGN TRANSLATION**" *by* **S. RANJITH KUMAR (160720747306), P. NIKITHA (160720747039) and P. LIKITHA (160720747302)** submitted in partial fulfillment of the requirements for the degree of Bachelor of Engineering in Artificial Intelligence and Data Science, of the Osmania University, Hyderabad, during the academic year 2023-2024, is a bonafide record of work carried out by them.

Dr. P. LAVANYA,
Professor &
Head of the Department.

Date:

Methodist College of Engineering andTechnology,
King Koti, Abids, Hyderabad-500001.

Department of Artificial Intelligence and Data Science

PROJECT APPROVAL CERTIFICATE

This is to certify that this Dissertation on project work entitled "**REAL TIME LANGUAGE ASSISTANT FOR SIGN TRANSLATION**" *by* **S. RANJITH KUMAR (160720747306), P. NIKITHA (160720747039) and P. LIKITHA (160720747302)** submitted in partial fulfillment of the requirements for the degree of Bachelor of Engineering in Artificial Intelligence and Data Science,of the Osmania University, Hyderabad, during the academic year 2023-2024, is a bonafide record of work carried out by them.

INTERNAL **EXTERNAL** **HOD**

ACKNOWLEDGEMENTS

We would like to express our sincere gratitude to my Project guide **Mrs. J. Sowmya, Assistant Professor, CSE,** for giving us the opportunity to work on this topic. It would never be possible for us to take this project to this level without her innovative ideas and her relentless support and encouragement.

We would like to thank our project coordinator **Dr. Diana Moses, Associate Professor ,Dept of CSE,** who helped us by being an example of high vision and pushing towards greater limits of achievement.

Our sincere thanks to **Dr. P. Lavanya, Professor** and **Head of the Department of Computer Science and Engineering,** for her valuable guidance and encouragement which has played a major role in the completion of the project and for helping us by being an example of high vision and pushing towards greater limits of achievement.

We would like to express a deep sense of gratitude towards the **Dr. Prabhu G. Benakop, Principal, Methodist College of Engineering and Technology,** for always being an inspiration and for always encouraging us in every possible way.

We would like to express a deep sense of gratitude towards the **Dr. Lakshmipathi Rao, Director, Methodist College of Engineering and Technology,** for always being an inspiration and for always encouraging us in every possible way.

We are indebted to the Department of Computer Science & Engineering and Methodist College of Engineering and Technology for providing us with all the required facility to carry our work in a congenial environment. We extend our gratitude to the CSE Department staff for providing us to the needful time to time whenever requested.

We would like to thank our parents for allowing us to realize our potential, all the support they have provided us over the years was the greatest gift anyone has ever given us and also for teaching us the value of hard work and education. Our parents have offered us with tremendous support and encouragement, thanks to our parents for all the moral support and the amazing opportunities they have given us over the years.

Vision & Mission

VISION

To become a leader in providing Computer Science & Engineering education with emphasis on knowledge and innovation.

MISSION

M1: To offer flexible programs of study with collaborations to suit industry needs

M2: To provide quality education and training through novel pedagogical practices

M3: To Expedite high performance of excellence in teaching, research and innovations.

M4: To impart moral, ethical valued education with social responsibility.

Program Educational Objectives

Graduates of Compute Science and Engineering at Methodist College of Engineering and Technology will be able to:

PEO1: Apply technical concepts, Analyze, synthesize data to Design and create novel products and solutions for the real-life problems.

PEO2: Apply the knowledge of Computer Science Engineering to pursue higher education with due consideration to the environment and society.

PEO3: Promote collaborative learning and spirit of team work through multidisciplinary projects

PEO4: Engage in life-long learning and develop entrepreneurial skills.

Program Specific Outcomes

At the end of 4 years, Compute Science and Engineering graduates at MCET will be able to:

PSO1: Apply the knowledge of Computer Science and Engineering in various domains like networking and data mining to manage projects in multidisciplinary environments.
PSO2: Develop software applications with open-ended programming environments**.**
PSO3: Design and develop solutions by following standard software engineering principles and implement by using suitable programming languages and platforms

PROGRAM OUTCOMES

PO1: Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2: Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO3: Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4: Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5: Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

PO6: The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7: Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8: Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9: Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10: Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11: Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12: Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change

ABSTRACT

Sign language is one of the oldest and most natural form of language for communication, but since most people do not know sign language and interpreters are very difficult to come by, we have come up with a real time method using neural networks for fingerspelling based on American sign language. In our method, the hand is first passed through a filter and after the filter is applied the hand is passed through a classifier which predicts the class of the hand gestures.

With the advancements in computer vision technology, learning and using sign languages to communicate with deaf and mute people has become easier. Exciting research is ongoing for providing a global platform for communication in different sign languages. In this project, we present a Deep Learning-based approach to recognize a sign performed in American Sign Language by capturing an image as input. The system can predict the signs of A to Z and Hand gestures performed by the user. By utilizing image processing to convert RGB data to grayscale images, an efficient reduction is achieved in the storage requirements and training time of the Convolutional Neural Network. The objective of the experiment is to find a mix of Image Processing and Deep Learning Architecture with lesser complexity to deploy the system.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1.INTRODUCTION

Speech impaired people use hand signs and gestures to communicate. Normal people face difficulty in understanding their language. Hence there is a need of a system which recognizes the different signs, gestures and conveys the information to the normal people. It bridges the gap between physically challenged people and normal people.

1.1 SIGN LANGUAGE AND SPEECH

Sign language is a visual language that is used by deaf people as their mother tongue. Unlike acoustically conveyed sound patterns, sign language uses body language and manual communication to fluidly convey the thoughts of a person. It is achieved by simultaneously combining hand shapes, orientation and movement of the hands, arms or body, and facial expressions. Sign languages are a complete, natural language that has the same linguistic properties as spoken languages, with grammar that differs from English. It is the primary language of those who are deaf and hard of hearing, and is used by many hearing people as well. There is no universal sign language. Different sign languages are used in different countries or regions. For example, British Sign Language (BSL) is a different language from ASL, and Americans who know ASL may not understand BSL. Some countries adopt features of ASL in their sign languages. No person or committee invented sign language. The exact beginnings of it are not clear, but some suggest that it arose more than 200 years ago from the intermixing of local sign languages and French Sign Language. Today's sign language includes some elements of LSF plus the original local sign languages; over time, these have melded and changed into a rich, complex, and mature language.

Modern sign language and modern LSF are distinct languages. While they still contain some similar signs, they can no longer be understood by each other's users. Sign Language is a language completely separate and distinct from English. It contains all the fundamental features of language, with its own rules for pronunciation, word formation, and word order. While every language 2 has ways of signalling different functions, such as asking a question rather than making a statement, languages differ in how this is done. For example, English speakers may ask a question by raising the pitch of their voices and by adjusting word order; Sign Language users ask a question by raising their eyebrows, widening their eyes, and tilting their bodies forward. Just as with other languages, specific ways of expressing ideas in sign language vary as much as users themselves. In addition to individual differences in expression, sign language has regional accents and dialects; just as certain English words are spoken differently in different parts of the country, sign language has regional variations in the rhythm of signing, pronunciation, slang, and signs used. Other sociological factors, including age and gender, can affect sign language usage and contribute to its variety, just as with spoken languages. Fingerspelling is part of sign language and is used to spell 3 out English words. In the finger spelled alphabet, each letter corresponds to a distinct handshape. Fingerspelling is often used for proper names or to indicate the English word for something.

1.2 PROJECT OVERVIEW

In this project, we will try to create an accurate model which would classify the sign language gestures using a computer's webcam and system should then translate these gestures into corresponding text and speech in real time by using a Convolutional Neural Network (CNN) to classify these gestures into relevant sign language classes and translate the recognized signs into spoken language (speech synthesis), where it focus on finger spelling sign language translation.

1.2.1 EXISTING SYSTEM

Not many have tried to accomplish this goal of translating sign language to speech. Those who tried have not succeeded in suggesting a good model. Analytics of various models were done but none of the studies thus far have succeeded in suggesting an accurate one.

1.2.2 PROPOSED SYSTEM

In this project, we will use image processing and neural networks to create an accurate model which would classify the input in whichever way necessary. Predefined data sets of sign language will be used for training our neural network model for classification. Image Processing will be used to take the images as input and for making them suitable for classification.

1.3 American Sign Language

American Sign Language (ASL) is a complete, natural language that has the same linguistic properties as spoken languages, with grammar that differs from English. ASL is expressed by movements of the hands and face. It is the primary language of many North Americans who are deaf and hard of hearing and is used by some hearing people as well. There is no universal sign language. Different sign languages are used in different countries or regions. For example, British Sign Language (BSL) is a different language from ASL, and Americans who know ASL may not understand BSL. Some countries adopt features of ASL in their sign languages. The exact beginnings of ASL are not clear, but some suggest that it arose more than 200 years ago from the intermixing of local sign languages and French Sign Language (LSF, or Langue des Signes Française). Today's ASL includes some elements of LSF plus the original local sign languages; over time, these have melded and changed into a rich, complex, and mature language. Modern ASL and modern LSF are distinct languages. While they still contain some similar signs, they can no longer be understood by each other's users. ASL is a language completely separate and distinct from English. It contains all the fundamental features of language, with its own rules for pronunciation, word formation, and word order. While every language has ways of signaling different functions, such as asking a

question rather than making a statement, languages differ in how this is done. For example, English speakers may ask a question by raising the pitch of their voices and by adjusting word order; ASL users ask a question by raising their eyebrows, widening their eyes, and tilting their bodies forward.

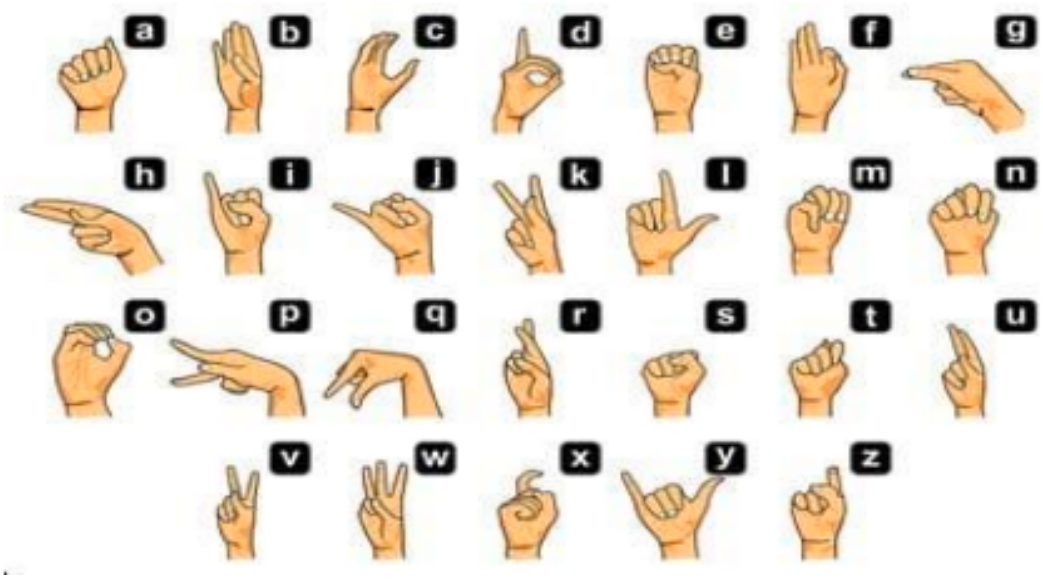


Fig 1: - American Sign Language

2.LITERATURE REVIEW

2.1 LITERATURE SURVEY

Author	Method	Dataset	Accuracy	Advantages or Disadvantage
Zafar Ahmed Ansari and Gaurav Harit, February 2016	K-means algorithm	Kinect sensor is used. RGB images of resolution 640 x 480 are captured along with their depth data.	Resultant accuracy was over 90%.	The main disadvantage is we were unable to use this dataset for training with convolutional neural networks.
Lionel Pigou et al	Convolutional Neural Network	The dataset includes a total of 20 distinct Italian gestures that have been signed by 27 people in different surroundings.	Resultant accuracy was over 91.7%	The main advantage is two steps were carried out – feature extraction and classifications of the actions.
Mathavan Suresh Anand, Nagarajan Mohan Kumar, Angappan Kumaresan, 2016	An Efficient Framework for Indian SignLanguage Recognition Using Wavelet Transform	The joint use of Discrete Wavelet Transform (DWT) based feature extraction and nearest neighbour classifier is used to recognize the sign language.	Resultant accuracy was over 99.23%	The main advantage is two important modules: feature extraction and classification.

Mandeep Kaur Ahuja, Amardeep Singh, July 2015	Hand Gesture Recognition Using PCA	driven hand gesture recognition based upon skin color model approach and thresholding approach along with an effective template matching with can be effectively used for human robotics applications and similar other applications	Resultant accuracy was over 95.23%	The main advantage is hand region is segmented by applying skin color model in YcbCr color space.
T. Bohra, S. Sompura, K. Parekh and P. Raut, 2019	Real-time two way sign language communication system built using image processing, deep learning and computer vision.	CNN model trained with a large dataset for 40 classes and was able to predict 17600 test images in 14 seconds	Resultant accuracy was over 99%	
H. Muthu Mariappan and V. Gomathi, 2019	Contour detection and fuzzy c-means algorithm	System was implemented on a dataset that contained videos recorded from 10 signers for several words and sentences.	Resultant accuracy was over 75%	The main advantage is Contours are used for detecting face, left and right hand. While fuzzy c-means algorithm is

				used to partition the input data into specified number of clusters
Kshitij Bantupalli and Ying Xie, 2018	CNN, LSTM and RNN	Various experiments were conducted with varying sample sizes and dataset consists of 100 different signs performed by 5 signers	Resultant accuracy was over 93%	The main advantage is A CNN model named Inception was used to extract spatial features from frames, LSTM for longer time dependencies and RNN to extract temporal features

Table 2.1 : Literature Survey

2.2 OBJECTIVES

- To develop an application which could be used by especially abled person to be able to convey their hand sign or gesture language into speech and aid an ordinary person to translate gesture to speech or hand sign language in order to make the communication more fluent.
- To be able to translate in the way that we would like to i.e. sign language to text or speech.
- Must be able to give continuous input to the model for it to classify.
- The output should be displayed with enough clarity to understand.

2.3 PROBLEM STATEMENT

Developing a neural network model that captures sign language gestures using a computer's webcam and system should then translate these gestures into corresponding text and speech in real time by using a Convolutional Neural Network (CNN).

3.DESIGN ANALYSIS

3.1 MODEL ARCHITECTURE

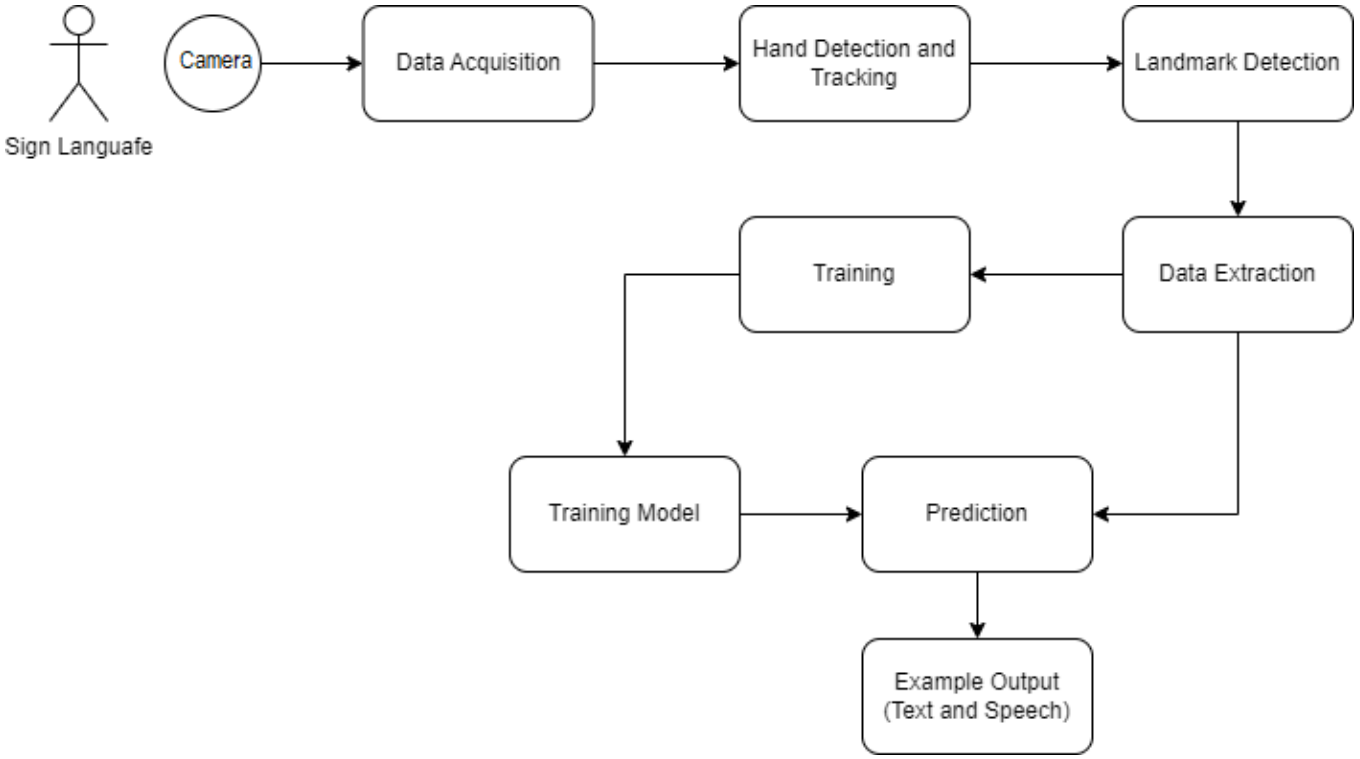


Figure 3.1 : System Framework

3.2 UML DIAGRAMS:

UML stands for Unified Modelling Language. UML is a standardized general-purpose modelling language in the field of object oriented software engineering. The standard is managed, and was created by, the Object Management Group. The goal is for UML to become a common language for creating models of object-oriented computer software. In its current form

UML is comprised of two major components: a Meta-model and a notation. In the future, some form of method or process may also be added to; or associated with, UML.

Unified Modelling Language is a standard language for specifying, Visualization, Constructing and documenting the artifacts of software system, as well as for business modelling and other non-software systems. UML represents a collection of best engineering practices that have proven successful in the modelling of large and complex systems. UML is a very important part of developing objects-oriented software and the software development process. UML uses mostly graphical notations to express the design of software projects.

GOALS:

The Primary goals in the design of UML are as follows:

1. Provide users a ready-to-use, expressive visual modelling Language so that developers can develop and exchange meaningful models.

2. Provide extendibility and specialization mechanisms to extend the core concepts.

3. Be independent of particular programming languages and development process.

4. Provide a formal basis for understanding the modelling language.

5. Encourage the growth of OO tools market.

3.2.1 USE CASE DIAGRAM:

A use case diagram in Unified Modelling Language (UML) is a type of behavioural diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.

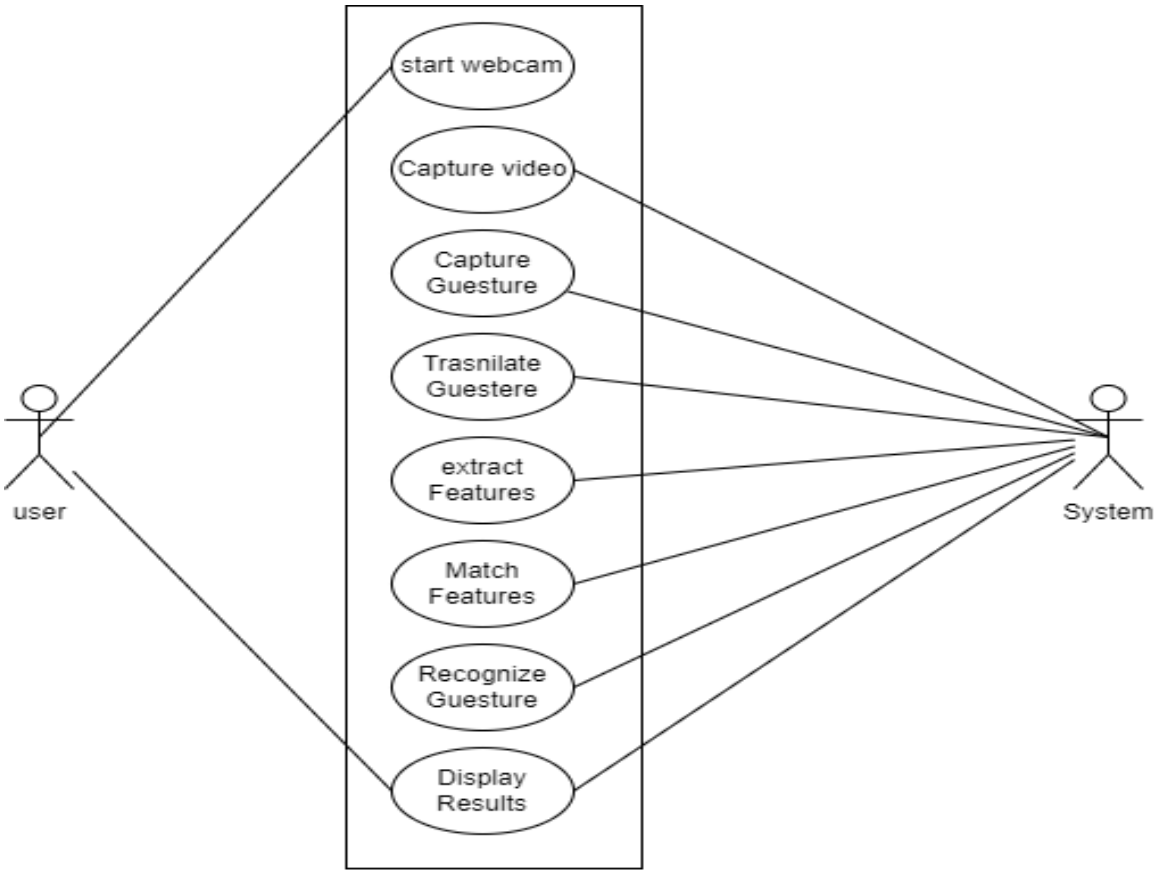


Figure 3.2 : Use Case Diagram

3.2.2 CLASS DIAGRAM:

Class diagram is a static diagram. It represents the static view of an application. Class diagram is not only used for visualizing, describing, and documenting different aspects of a system but also for constructing executable code of the software application. The purpose of class diagram is to model the static view of an application. Class diagrams are the only diagrams which can be directly mapped with object-oriented languages and thus widely used at the time of construction.

UML diagrams like activity diagram, sequence diagram can only give the sequence flow of the application, however class diagram is a bit different. It is the most popular UML diagram in the coder community. The purpose of the class diagram can be summarized as − -Analysis and design of the static view of an application

-Describe responsibilities of a system.

-Base for component and deployment diagrams.

-Forward and reverse engineering.

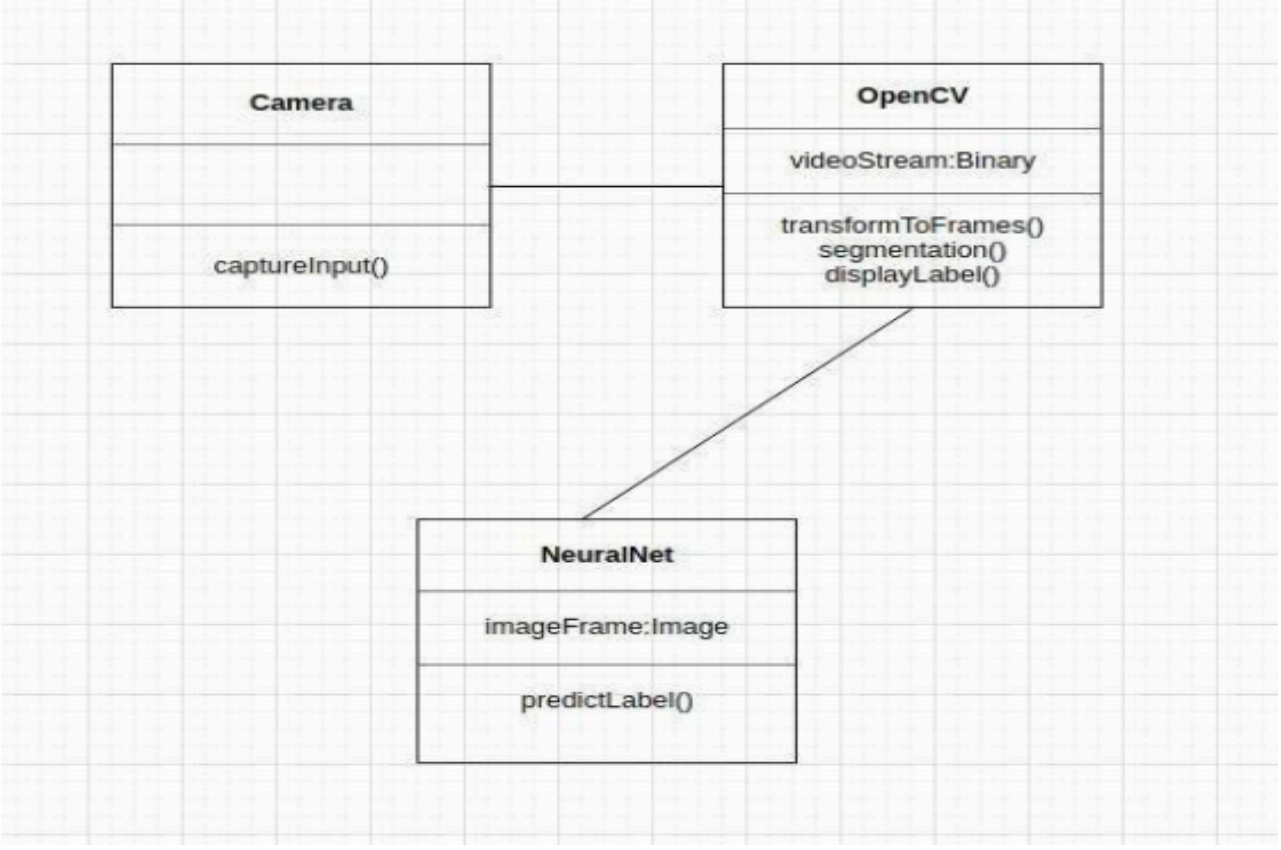


Figure 3.3 : Class Diagram

3.2.3 SEQUENCE DIAGRAM:

A sequence diagram is the most commonly used interaction diagram. An interaction diagram is used to show the interactive behavior of a system. Since visualizing the interactions in a system can be a cumbersome task, we use different types of interaction diagrams to capture various features and aspects of interaction in a system.

A sequence diagram simply depicts interaction between objects in a sequential order i.e., the order in which these interactions take place. We can also use the terms event diagrams or event scenarios to refer to a sequence diagram. Sequence diagrams describe how and in what order the objects in a system function. These diagrams are widely used by businessmen and software developers to document and understand requirements for new and existing

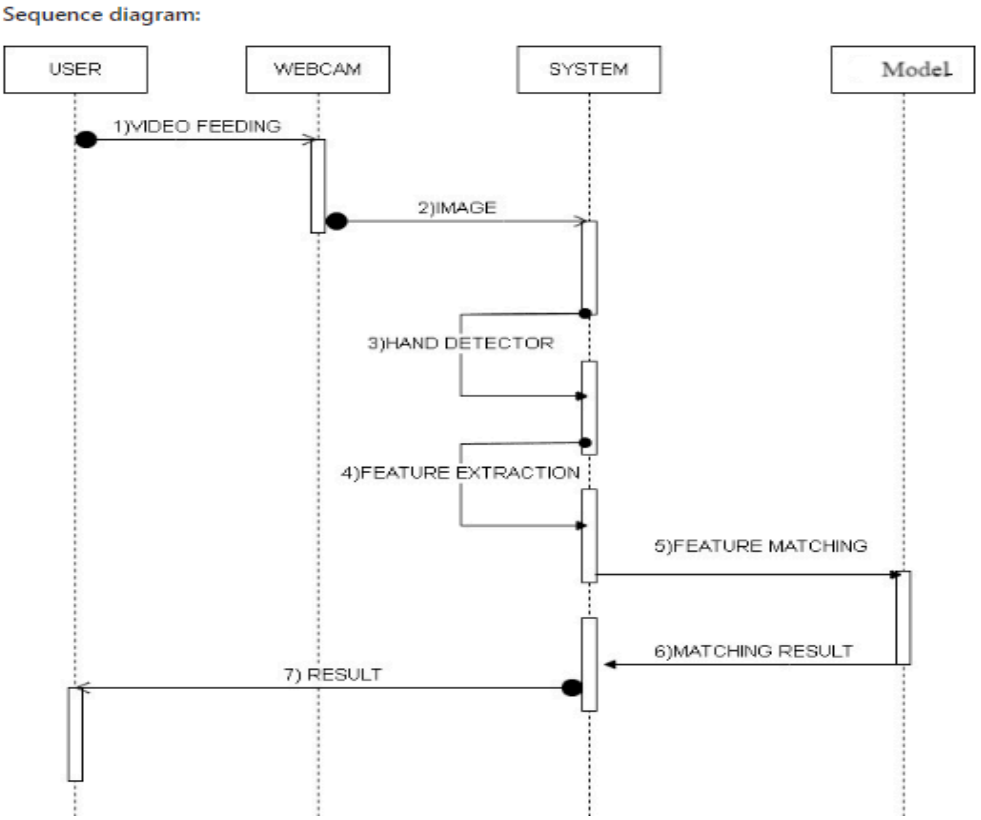


Figure 3.4 : Sequence Diagram

3.3 Dataflow Diagram

The DFD is also known as bubble chart. It is a simple graphical formalism that can be used to represent a system in terms of the input data to the system, various processing carried out on these data, and the output data is generated by the system. It maps out the flow of information for any process or system, how data is processed in terms of inputs and outputs. It uses defined symbols like rectangles, circles and arrows to show data inputs, outputs, storage points and the routes between each destination. They can be used to analyse an existing system or model of a new one. A DFD can often visually —sayǁ things that would be hard to explain in words and they work for both technical and non- technical. There are four components in DFD:

1. External Entity
2. Process
3. Data Flow
4. data Store

1. External Entity:

It is an outside system that sends or receives data, communicating with the system. They are the sources and destinations of information entering and leaving the system. They might be an outside organization or person, a computer system or a business system. They are known as terminators, sources and sinks or actors. They are typically drawn on the edges of the diagram. These are sources and destinations of the system‘s input and output.

Representation:

2. Process:

It is just like a function that changes the data, producing an output. It might perform computations for sort data based on logic or direct the dataflow based on business rules

Representation:

3.Data Flow:

A dataflow represents a package of information flowing between two objects in the data-flow diagram, Data flows are used to model the flow of information into the system, out of the system and between the elements within the system.

Representation:

4. Data Store:

These are the files or repositories that hold information for later use, such as a database table or a membership form. Each data store receives a simple label.

Representation:

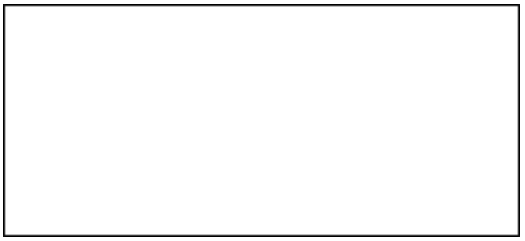


Fig 3.5 : Dataflow Diagram for Sign Language Recognition - Level 1

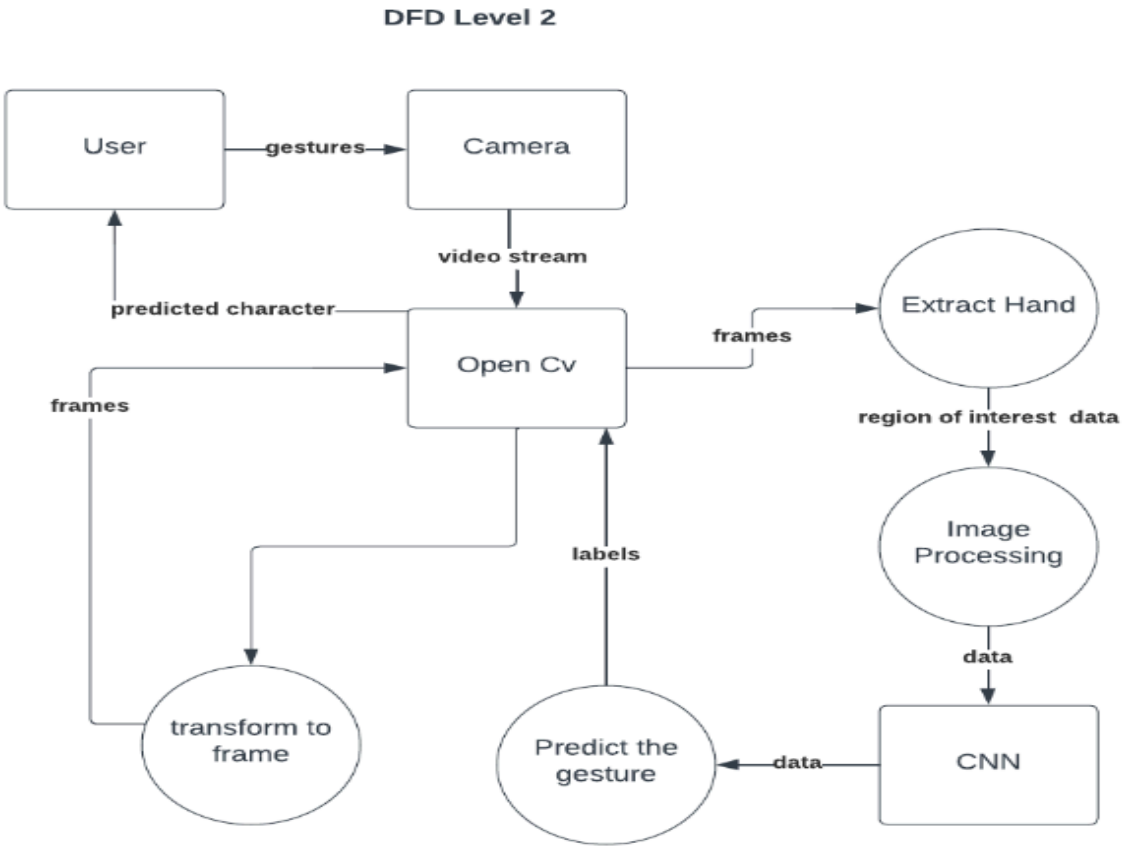


Fig 3.6 :Dataflow Diagram for Sign Language Recognition - Level 2

4.MODULES AND IMPLEMENTATION

4.1 Data Acquisition

The data acquisition is one step of the sign language recognition, this step consists of digitize the information from the user to make it usable to classification algorithms, the information from the user can be acquired in different ways depending on the tools used, such as visual based like cameras.

This is a primary and essential step in sign recognition whole process. Camera interfacing unnecessary task to capture images with the help of Webcam. Now a days lots of Laptops are coming with inbuilt camera system so that's helps lot for capturing images to process it further. Gestures can be captured by inbuilt camera to detect hand movements and position. Capturing 30fps will be sufficient to process images; more input images may lead to higher computational time and will make system slow and vulnerable.

4.1.1 Getting the Dataset:

Datasets of predefined sign language was taken from Kaggle. It had approximately 87,000 images which are 200X200 pixels belonging to 29 classes of which 26 are for the letters A-Z and 3 classes for *SPACE*, *DELETE* and *NOTHING*. These 3 classes are very helpful in real-time applications, and classification.corresponding to each alphabet including both training and test data sets.

Out[4]:

	label	path
0	N	/kaggle/input/asl-alphabet/asl_alphabet_train/...
1	N	/kaggle/input/asl-alphabet/asl_alphabet_train/...
2	N	/kaggle/input/asl-alphabet/asl_alphabet_train/...
3	N	/kaggle/input/asl-alphabet/asl_alphabet_train/...
4	N	/kaggle/input/asl-alphabet/asl_alphabet_train/...
...	...	...
86995	J	/kaggle/input/asl-alphabet/asl_alphabet_train/...
86996	J	/kaggle/input/asl-alphabet/asl_alphabet_train/...
86997	J	/kaggle/input/asl-alphabet/asl_alphabet_train/...
86998	J	/kaggle/input/asl-alphabet/asl_alphabet_train/...
86999	J	/kaggle/input/asl-alphabet/asl_alphabet_train/...

87000 rows × 2 columns

Fig 4.1 Data set Details

4.2 Data Pre-Processing

Data preprocessing is an important step to prepare the data to form a model. There are many important steps in data preprocessing, such as data cleaning, data transformation, and feature selection. Data cleaning and transformation are methods used to remove outliers and standardize the data so that they take a form that can be easily used to create a model. This section focuses mostly on the applications of models and will emphasize the feature selection portion of data preprocessing. When trying to create a model, a data set may contain hundreds of variables (descriptors); however, many of these variables will contain redundant data. In order to simplify the dimensionality of the model, it is important to select only variables that contain unique and important information.

The training data of the sign images was taken and was cleaned to make it ready for classification. All the images were appended to an image array. The image array was then converted to a numpy array. The array was normalized by converting the values to 'float32' and dividing it with the total sum of the pixel values i.e. 255. The labels corresponding to each image was stored in a array as well and was converted to categorical values using the internal function present in keras module. The whole image data was split into training and testing using the internal function present in scikit-learn module. The shape of the train and test data was printed and returned back for the model to consume.

- **Background subtraction (BS)** is a common and widely used technique for generating a foreground mask (namely, a binary image containing the pixels belonging to moving objects in the scene) by using static cameras. As the name suggests, BS calculates the foreground mask performing a subtraction between the current frame and a background model, containing the static part of the scene or, more in general, everything that can be considered as background given the characteristics of the observed scene..
- **GrayScale Conversion** An image is a collection of pixels. Pixels are arranged to form an image. Size of an image is represented in matrix form as IxJ, where i is the number of pixels in row and j is the number of pixels in column. Each pixel has its pixel values with three values: Red, Green and Blue. Grayscale conversion means converting an image into gray representation, omitting other colours. This can be done by applying mathematical formulation on the RGB values and replacing them in the image or creating new image with those new values.

4.2.1 Feature Extraction

Feature extraction refers to the process of transforming raw data into numerical features that can be processed while preserving the information in the original data set. It yields better results than applying machine learning directly to the raw data.

Automated feature extraction uses specialized algorithms or deep networks to extract features automatically from signals or images without the need for human intervention. This technique can be very useful when you want to move quickly from raw data to developing machine learning algorithms.

- Gaussian filter is used as a pre-processing technique to make the image smooth and eliminate all the irrelevant noise.

- Intensity is analyzed and Non-Maximum suppression is implemented to remove false edges.
- For better pre-processed image data, double thresholding is implemented to consider only the strong edges in the images.
- All the weak edges are finally removed and only the strong edges are considered for the further phases.

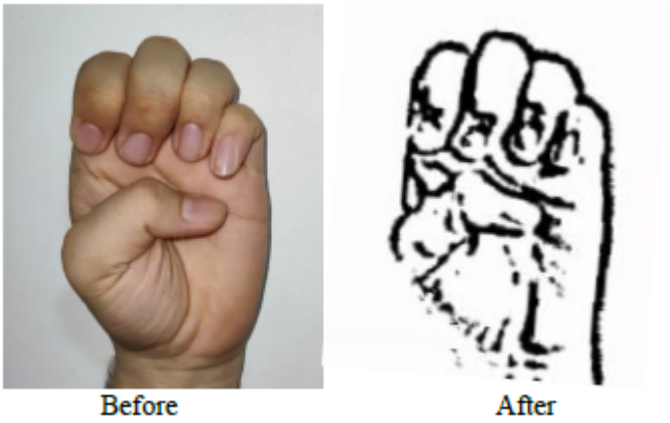


Fig 4.2: Pre-processed image with extracted features

4.3 Dataset Splitting

Data splitting involves dividing a dataset into training, validation, and testing subsets. Data splitting divides a dataset into three main subsets: the training set, used to train the model; the validation set, used to track model parameters and avoid overfitting; and the testing set, used for checking the model's performance on new data. Each subset serves a unique purpose in the iterative process of developing a machine-learning model.

The Training Set It is the set of data that is used to train and make the model learn the hidden features/patterns in the data. In each epoch, the same training data is fed to the neural network architecture repeatedly, and the model continues to learn the features of the data. The training set should have a diversified set of inputs so that the model is trained in all scenarios and can predict any unseen data sample that may appear in the future.

The Validation Set The validation set is a set of data, separate from the training set, that is used to validate our model performance during training. This validation process gives information that helps us tune the model's hyperparameters and configurations accordingly. It is like a critic telling us whether the training is moving in the right direction or not. The model is trained on the training set, and, simultaneously, the model evaluation is performed on the validation set after every epoch.

The main idea of splitting the dataset into a validation set is to prevent our model from overfitting i.e., the model becomes really good at classifying the samples in the training set but cannot generalize and make accurate classifications on the data it has not seen before.

The Test Set is a separate set of data used to test the model after completing the training. It provides an unbiased final model performance metric in terms of accuracy, precision, etc. Putting 80% of the data in the training set, 10% in the validation set, and 10% in the test set is a good split to start with, if the validation set is too small, then the evaluation metrics like accuracy, precision, recall, and F1 score will have large variance and will not lead to the proper tuning of the model.

```
#Train/test split
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X_data, y_data, test_size=0.3,random_state=42,stratify=y_data)
```

Fig-4.3: Splitting of Data

This above code snippet describes the train_test_split function from scikit-learn to partition a dataset into separate training and testing subsets.

Upon execution: The feature data **X_data** and target data **y_data** are passed as input to the function. By specifying **test_size=0.3**, 30% of the dataset is reserved for testing, while the remaining 70% is allocated for training. The **random_state=42** parameter ensures the reproducibility of the split, meaning the same random split will be generated each time the code is run. Setting **stratify=y_data** maintains the distribution of class labels in both the training and testing sets, which is particularly useful for imbalanced datasets. The function returns four datasets: **X_train** containing features for training, **X_test** containing features for testing, **y_train** containing corresponding labels for training, and **y_test** containing corresponding labels for testing.

This partitioning enables model training on the training set and subsequent evaluation of the model's performance on unseen data from the testing set.

4.4 CONVOLUTIONAL NEURAL NETWORK

4.4.1 Building Convolutional model

Creating CNN Model: After pre-processing of the training data, we will create a CNN model which is used for classification. Keras module is used extensively here to create the model.

Input Layer: The input layer expects images with dimensions of 64x64 pixels and 3 color channels (RGB). It serves as the entry point for the neural network, where the raw image data is fed into the model for processing.

Convolutional Layers: This layer consists of 32 filters, each with a kernel size of (5, 5). These filters convolve across the input image, extracting features such as edges and textures. The ReLU activation function is applied element-wise to introduce non-linearity, allowing the model to learn complex patterns. MaxPooling is performed with a pool size of (2, 2) to downsample the feature maps by selecting the maximum value within each pooling window.

Convolutional Layer 2: Another convolutional layer is added with 64 filters, each with a smaller kernel size of (3, 3). This allows the network to learn more intricate features at a finer scale. Similar to the first convolutional layer, ReLU activation and MaxPooling are applied to the feature maps.

Convolutional Layer 3: A third convolutional layer is included, also with 64 filters and a kernel size of (3, 3). This layer further extracts high-level features from the feature maps generated by the previous layers. ReLU activation and MaxPooling are applied again for non-linearity and downsampling.

Flatten Layer:The Flatten layer transforms the 3D feature maps into a one-dimensional vector, preparing the data for input into the dense layers. This flattening operation retains the spatial structure of the features while converting them into a format suitable for fully connected layers.

Dense Layer 1: Following the Flatten layer, a dense layer with 128 neurons is added. This fully connected layer allows the network to learn complex relationships between the extracted features. The ReLU activation function is applied to introduce non-linearity.

Output Layer: The final layer is a dense layer with 29 neurons, corresponding to the number of classes for classification. The softmax activation function is used to compute the probability distribution over the classes, providing the model's prediction for each class.

Activation Function: We have used ReLu (Rectified Linear Unit) in each of the layers (convolutional as well as fully connected neurons). ReLu calculates max(x,0) for each input pixel. This adds nonlinearity to the formula and helps to learn more complicated features. It helps in removing the vanishing gradient problem and speeding up the training by reducing the computation time. At the last activation function, we used SOFTMAX function. It is used as the activation function in the output layer of neural network models that predict a

multinomial probability distribution. That is, SoftMax is used as the activation function for multi-class classification problems where class membership is required on more than two class labels.

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 60, 60, 32)	2,432
activation (Activation)	(None, 60, 60, 32)	0
max_pooling2d (MaxPooling2D)	(None, 30, 30, 32)	0
conv2d_1 (Conv2D)	(None, 28, 28, 64)	18,496
activation_1 (Activation)	(None, 28, 28, 64)	0
max_pooling2d_1 (MaxPooling2D)	(None, 14, 14, 64)	0
conv2d_2 (Conv2D)	(None, 12, 12, 64)	36,928
activation_2 (Activation)	(None, 12, 12, 64)	0
max_pooling2d_2 (MaxPooling2D)	(None, 6, 6, 64)	0
flatten (Flatten)	(None, 2304)	0
dense (Dense)	(None, 128)	295,040
dense_1 (Dense)	(None, 29)	3,741

Total params: 356,637 (1.36 MB)

Trainable params: 356,637 (1.36 MB)

Non-trainable params: 0 (0.00 B)

Fig:4.4 : Model Summary

4.4.2 Model Compilation

Compilation Process: The ‘compile’ method configures the model for training by specifying the optimizer, loss function, and evaluation metrics.

Optimizer: 'adam' is chosen as the optimizer. Adam is an adaptive learning rate optimization algorithm that is widely used for training neural networks. It adapts the learning rate during training, which can lead to faster convergence and better performance compared to traditional optimization methods.Adam combines the benefits of AdaGrad and RMSProp optimization algorithms. Adam combines the benefits of AdaGrad and RMSProp optimization algorithms.

Loss Function: The loss function is set to 'categorical_crossentropy'. This is a common choice for multi-class classification problems where each example belongs to a single class. It measures the difference between the true distribution and the predicted distribution of the classes, aiming to minimize this difference during training. The goal is to minimize this difference during training by adjusting the model's parameters.

Metrics: The model's performance during training and evaluation will be monitored using the 'accuracy' metric. Accuracy measures the proportion of correctly classified examples out of all examples. A higher accuracy indicates better performance, but it may not be sufficient for imbalanced datasets

```
#compiling
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

Fig 4.5 : Compiling model

4.4.3 Model Training

After the model is created, we need to fit our train data into it so that the model will learn about the data through training and then it would be used for further classifying real-time data. We use the "fit" method to give our training data. The batch_size is set as 64 which means that images will be fed to the CNN model in batches with 64 each in one batch. The epoch is set to be 5 i.e. for every five minutes the data will be fed to the model. After the training is done, an object is returned which has all the metrics that we had put while creating the model.

The **model.fi**t method initiates the training process for the neural network model. It takes the training data (X_train) and corresponding target labels (y_cat_train) as input to train the model.

Epochs (epochs=5): An epoch refers to one complete pass through the entire training dataset during model training. Setting epochs=5 specifies that the training process will iterate over the entire dataset 5 times. Increasing the number of epochs may allow the model to learn more complex patterns from the data, but it could also lead to overfitting if not carefully monitored.

Batch Size (batch_size=64): The batch size determines the number of samples processed before the model's parameters are updated during training. With batch_size=64, the training data is divided into batches, each containing 64 samples. Using mini-batches (as opposed to processing the entire dataset at once) can help improve the efficiency of training and utilize parallel computing resources.

Verbose (verbose=2): The verbose parameter controls the amount of information printed during training. Setting verbose=2 prints a progress bar for each epoch, showing the training progress and performance metrics such as loss and accuracy. It provides useful feedback on the training process, especially when training large models over many epochs.

Validation Data (validation_data=(X_test, y_cat_test)): Validation data, consisting of features (X_test) and corresponding target labels (y_cat_test), is used to evaluate the model's performance during training. After each epoch, the model's performance on the validation data is computed, allowing for early stopping and monitoring for overfitting. It helps to ensure that the model generalizes well to unseen data and does not memorize the training examples.

Callbacks ([early_stop]): Callbacks are functions that are called at specific points during training, allowing for custom actions such as early stopping or model checkpointing. the early_stop callback is passed to the callbacks parameter, enabling early stopping based on the validation loss. If the validation loss does not improve for a specified number of epochs, training will stop early to prevent overfitting.

```python
# Model fitting
model.fit(X_train, y_cat_train,
          epochs=5,
          batch_size=64,
          verbose=2,
          validation_data=(X_test, y_cat_test),
         callbacks=[early_stop])
```

```
Epoch 1/5
952/952 - 206s - 217ms/step - accuracy: 0.9907 - loss: 0.0305 - val_accuracy: 0.9871 - val_loss: 0.0480
Epoch 2/5
952/952 - 254s - 267ms/step - accuracy: 0.9926 - loss: 0.0259 - val_accuracy: 0.9926 - val_loss: 0.0208
Epoch 3/5
952/952 - 196s - 206ms/step - accuracy: 0.9917 - loss: 0.0264 - val_accuracy: 0.9934 - val_loss: 0.0222
Epoch 4/5
952/952 - 202s - 212ms/step - accuracy: 0.9939 - loss: 0.0210 - val_accuracy: 0.9928 - val_loss: 0.0251

<keras.src.callbacks.history.History at 0x7b0ef656a8c0>
```

Fig 4.6 : Model Training

The model metrics are

	accuracy	loss	val_accuracy	val_loss
0	0.990706	0.030532	0.987126	0.048018
1	0.992562	0.025880	0.992605	0.020807
2	0.991675	0.026411	0.993410	0.022225
3	0.993859	0.021045	0.992835	0.025068

Figure 4.7 : Model Evaluation Metrics

4.5 Model Evaluation

Evaluation metrics can help you assess your model's performance, monitor your ML system in production, and control your model to fit your business needs. Our goal is to create and select a model which gives high accuracy on out-of-sample data. It's very crucial to use multiple evaluation metrics to evaluate your model because a model may perform well using one measurement from one evaluation metric while may perform poorly using another measurement from another evaluation metric.

4.5.1 ACCURACY

Accuracy of an algorithm is represented as the ratio of correctly classified predictions (TP+TN) to the total number of predictions (TP+TN+FP+FN).

$$\text{Accuracy} = (TP+TN) / (TP+TN+FP+FN)$$

4.5.2 PRECISION

Precision is the ability of a classifier not to label an instance positive that is actually negative. For each class it is defined as the ratio of true positives to the sum of true and false positives.

- TP – True Positives
- FP – False Positives

Precision: Accuracy of positive predictions.

$$\text{Precision} = TP / (TP+FP)$$

4.5.3 RECALL

Recall is the ability of a classifier to find all positive instances. For each class it is defined as the ratio of true positives to the sum of true positives and false negatives.

Recall: Fraction of positives that were correctly identified.

$$\text{Recall} = TP / (TP+FN)$$

4.5.4 F1 SCORE

The F1 score is a weighted harmonic mean of precision and recall such that the best score is 1.0 and the worst is 0.0. Generally speaking, F1 scores are lower than accuracy measures as they embed precision and recall into their computation. As a rule of thumb, the weighted average of F1 should be used to compare classifier models, not global accuracy.

$$\text{F1 Score} = 2*(\text{Recall}*\text{precision}) / (\text{Recall}+\text{Precision})$$

F1 score is having equal relative contribution of precision and recall.

4.5.5 SUPPORT

Support may be defined as the number of samples of the true response that lies in each class of target values.

We can use the classification_report function of *sklearn.metrics* to get the classification report of our classification model.

	precision	recall	f1-score	support
0	1.00	1.00	1.00	900
1	1.00	0.97	0.98	900
2	1.00	1.00	1.00	900
3	1.00	1.00	1.00	900
4	0.99	0.99	0.99	900
5	1.00	1.00	1.00	900
6	1.00	0.97	0.99	900
7	1.00	0.99	0.99	900
8	0.99	1.00	0.99	900
9	0.99	1.00	1.00	900
10	0.99	0.96	0.98	900
11	0.99	1.00	1.00	900
12	1.00	0.99	0.99	900
13	0.99	1.00	0.99	900
14	1.00	0.99	1.00	900
15	0.98	1.00	0.99	900
16	1.00	1.00	1.00	900
17	0.97	0.98	0.98	900
18	0.99	0.99	0.99	900
19	1.00	0.99	0.99	900
20	0.98	0.98	0.98	900
21	0.95	0.99	0.97	900
22	0.99	1.00	1.00	900
23	1.00	1.00	1.00	900
24	1.00	1.00	1.00	900
25	1.00	1.00	1.00	900
26	1.00	1.00	1.00	900
27	1.00	1.00	1.00	900
28	1.00	1.00	1.00	900
accuracy			0.99	26100
macro avg	0.99	0.99	0.99	26100
weighted avg	0.99	0.99	0.99	26100

Figure 4.8 : Evaluation Matrix of Model

4.6 Creating a GUI using Tkinter

Tkinter is a Python library that serves as the standard GUI (Graphical User Interface) toolkit for Python. It allows developers to create desktop applications with graphical interfaces in a straightforward and efficient manner. Tkinter is included with most Python installations, making it readily available for development without the need for additional installations.

Enchant a versatile and powerful spellchecking library for Python that goes beyond traditional spellchecking to offer a comprehensive suite of text processing tools. In GUI building we used enchant, It's a sentence spellchecking library for Python that provides developers with the tools to incorporate spellchecking functionality into their applications seamlessly. It offers support for multiple languages and dictionaries **spell_checker = enchant.Dict("en_US")**

Enchant used in Suggestion Generation, Grammar Checking and Correction, Spellchecking Across Languages, Customization and Extensibility, Integration with Python Ecosystem.

We created a graphical user interface (GUI) application using the Tkinter library for sign language classification and text-to-speech conversion.

Importing Libraries: The begins by importing necessary libraries, including Tkinter, PIL (Python Imaging Library) for image processing, OpenCV (cv2) for computer vision tasks, and pyttsx3 for text-to-speech conversion.

Loading Pre-trained Model: It loads a pre-trained deep learning model (`model.h5`) for sign language classification using Keras.

Initializing Text-to-Speech Engine and Spell Checker: The initializes a text-to-speech engine and a spell checker using the pyttsx3 and enchant libraries, respectively.

Defining Functions:

- `sign_to_text`: Converts sign language frames to text using the loaded model.
- `text_to_speech`: Converts text to speech using the text-to-speech engine.
- `start_video_feed` and `stop_video_feed`: Functions to start and stop the video feed from the webcam.
- `clear_text`: Clears the predicted text.
- `convert_text_to_speech`: Reads text from the textbox and converts it to speech.

Initializing Main Window: The main window is created using Tkinter's `Tk` class, and a notebook widget is added to hold tabs.

Creating Classification Tab:

- The classification tab contains:
  - A frame for displaying the video feed from the webcam.
  - A label to display the classified text.
  - A textbox to input sentences.

- Buttons to start/stop the video feed, clear text, and convert text to speech.

Initializing ROI Window: A separate window (`roi_window`) is created to display the region of interest (ROI) frame for sign language classification.

Updating Frames: The `update_frame` function continuously captures frames from the webcam, classifies sign language gestures, updates the GUI components, and displays the ROI frame.

Starting Main Loop: The `root.mainloop()` function starts the Tkinter event loop, allowing the GUI application to run.

This provides an interactive interface for sign language classification and text-to-speech conversion, demonstrating the integration of computer vision, deep learning, and natural language processing techniques within a Tkinter-based application.

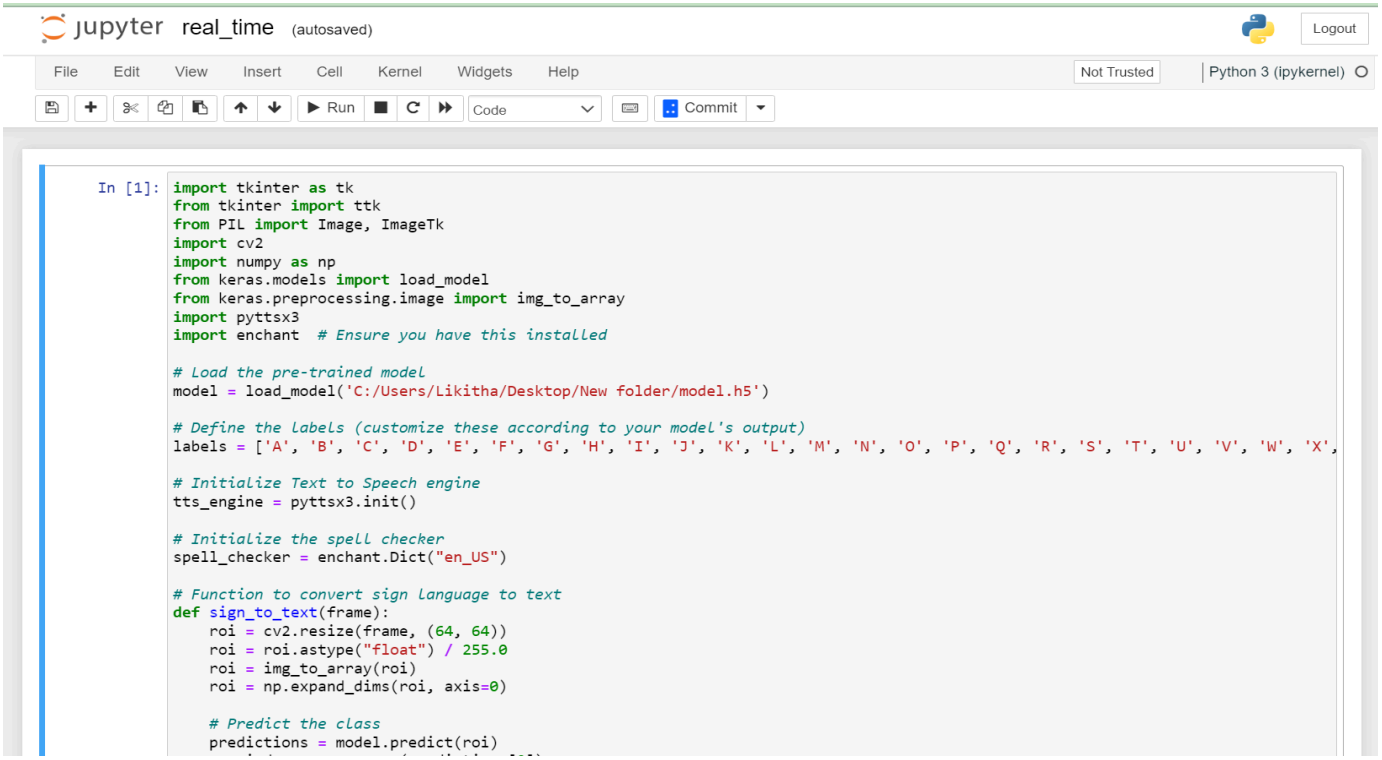


Fig 4.9 : Source code of Real_Time GUI Generation

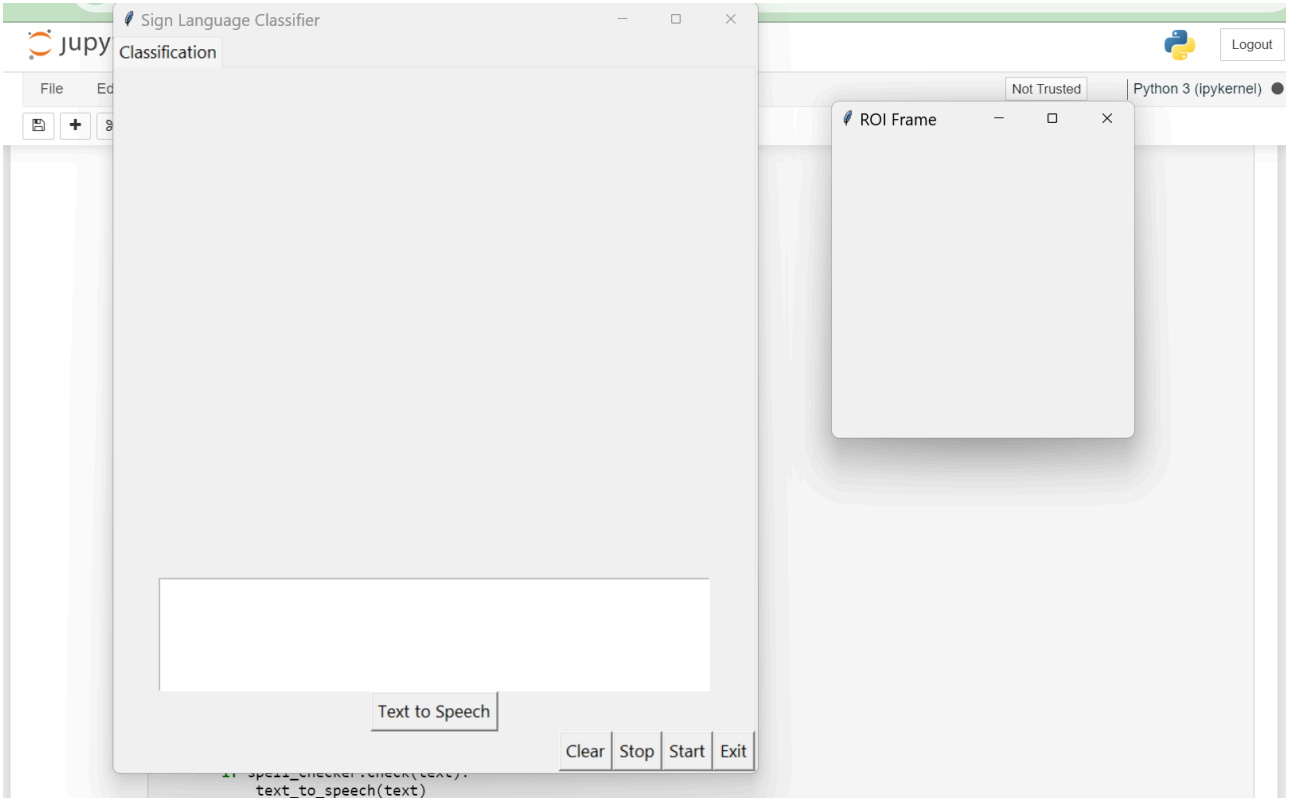


Fig 4.10 GUI OUTPUT

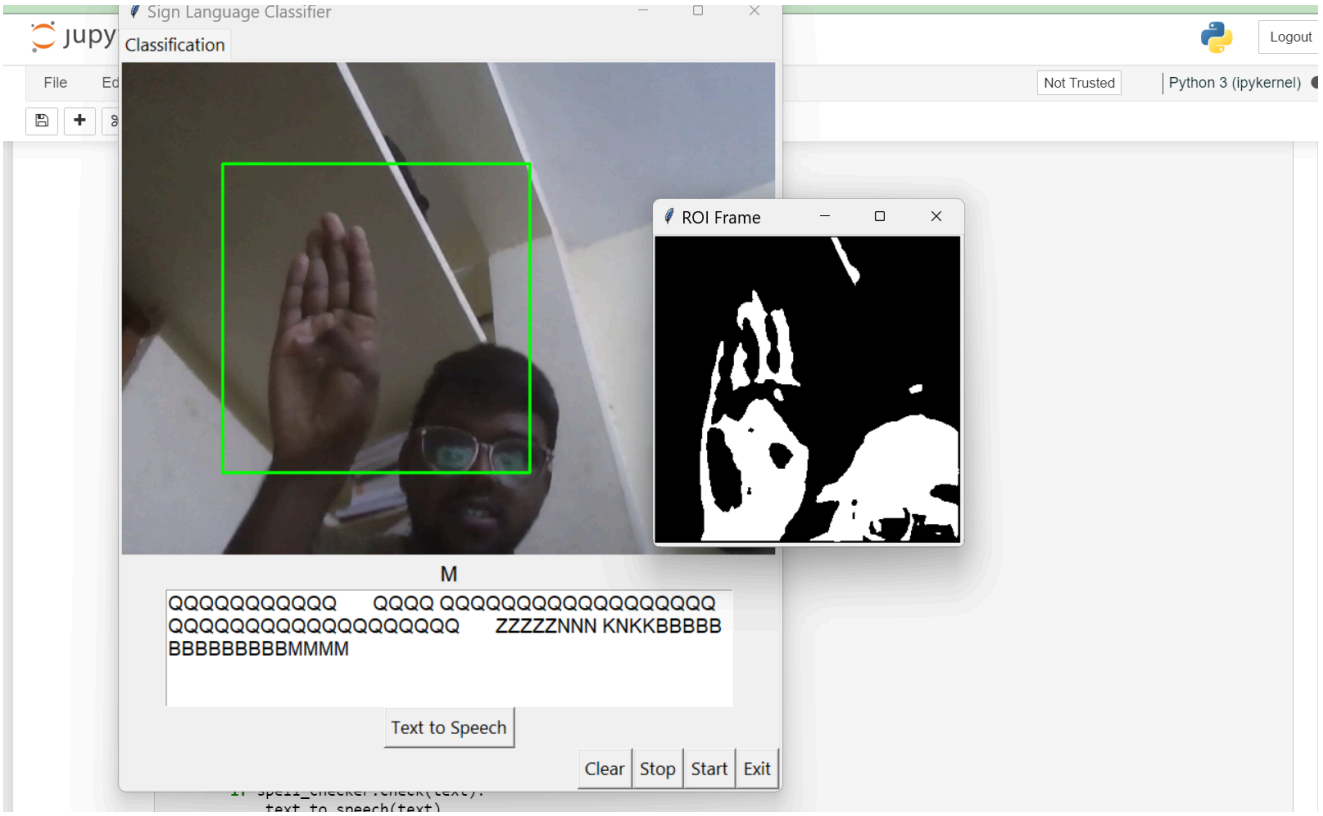


Fig 4.11 : Sign to Speech

5. TESTING

5.1 SYSTEM TESTING

The purpose of testing is to discover errors. Testing is a process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub assemblies, assemblies and/or a finished product. It is the process of exercising software with the intent of ensuring that the software system meets its requirements and user expectations and does not fail in an unacceptable manner.

Software testing is an important element of the software quality assurance and represents the ultimate review of specification, design and coding. The increasing feasibility of software as a system and the cost associated with the software failures are motivated forces for well planned through testing.

5.1.1 Testing Objectives:

There are several rules that can serve as testing objectives they are:

- Testing is a process of executing a program with the intent of finding an error.
- A good test case is the one that has a high probability of 39 finding an undiscovered error.

5.2 Types of Testing:

In order to make sure that the system does not have errors, the different levels of testing strategies that are applied at different phases of software development are :

5.2.1 Unit Testing:

Unit testing is done on individual models as they are completed and becomes executable. It is confined only to the designer's requirements. Unit testing is different from and should be preceded by other techniques, including:

- Inform Debugging
- Code Inspection

5.2.2 Black Box testing

In this strategy some test cases are generated as input conditions that fully execute all functional requirements for the program. This testing has been used to find error in the following categories: Incorrect or missing functions

- Interface errors

- Errors in data structures are external database access
- Performance error
- Initialisation and termination of errors
- In this testing only the output is checked for correctness
- The logical flow of data is not checked.

5.2.3 White Box testing

In this the test cases are generated on the logic of each module by drawing flow graphs of that module and logical decisions are tested on all the cases.

It has been used to generate the test cases in the following cases:

- Guarantee that all independent paths have been executed.
- Execute all loops at their boundaries and within their operational bounds.
- Execute internal data structures to ensure their validity.

5.2.4 Integration Testing

Integration testing ensures that software and subsystems work together a whole. It test the interface of all the modules to make sure that the modules behave properly when integrated together. It is typically performed by developers, especially at the lower, module to module level. Testers become involved in higher levels.

5.2.5 System Testing

Involves in house testing of the entire system before delivery to the user. The aim is to satisfy the user that the system meets all requirements of the client‘s specifications. It is conducted by the testing organization if a company has one. Test data may range from and generated to production.

Requires test scheduling to plan and organize:

- Inclusion of changes/fixes.
- Test data to use

One common approach is graduated testing: as system testing progresses and (hopefully) fewer and fewer defects are found, the code is frozen for testing for increasingly longer time periods.

5.2.6 Acceptance Testing

It is a pre-delivery testing in which the entire system is tested at the client's site on real world data to find errors.

User Acceptance Test (UAT) -

"Beta testing" : Acceptance testing in the customer environment.

Requirements traceability:

- Match requirements to test cases.
- Every requirement has to be cleared by at least one test case.
- Display in a matrix of requirements vs test cases.

Test Results: All the test cases mentioned above passed successfully. No defects encountered.

```
train_folder = '/kaggle/input/asl-alphabet/asl_alphabet_train/asl_alphabet_train'
all_data = []
for folder in os.listdir(train_folder):

    label_folder = os.path.join(train_folder, folder)
    onlyfiles = [{'label':folder,'path':os.path.join(label_folder, f)} for f in os.listdir(label_folder) if os.path.isfile(os.pat
    #print(onlyfiles)
    all_data += onlyfiles
data_df = pd.DataFrame(all_data)
data_df
```

	label	path
0	N	/kaggle/input/asl-alphabet/asl_alphabet_train/...
1	N	/kaggle/input/asl-alphabet/asl_alphabet_train/...
2	N	/kaggle/input/asl-alphabet/asl_alphabet_train/...
3	N	/kaggle/input/asl-alphabet/asl_alphabet_train/...
4	N	/kaggle/input/asl-alphabet/asl_alphabet_train/...
...	...	...
86995	J	/kaggle/input/asl-alphabet/asl_alphabet_train/...
86996	J	/kaggle/input/asl-alphabet/asl_alphabet_train/...
86997	J	/kaggle/input/asl-alphabet/asl_alphabet_train/...
86998	J	/kaggle/input/asl-alphabet/asl_alphabet_train/...
86999	J	/kaggle/input/asl-alphabet/asl_alphabet_train/...

87000 rows × 2 columns

Fig 5.1 : Loading Dataset

```
class_names = list(train_generator.class_indices.keys())
print(class_names)
```

```
['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U
'Z', 'del', 'nothing', 'space']
```

Fig 5.2 : Listing of files and counting of files in directory

```python
from tensorflow.keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(
    rescale=1./255,
)
train_generator = train_datagen.flow_from_directory(
    '/kaggle/input/asl-alphabet/asl_alphabet_train/asl_alphabet_train',
    target_size=(400, 400),
    batch_size=32,
    class_mode='categorical',
    color_mode='grayscale'
)
```

```
Found 87000 images belonging to 29 classes.
```

Fig 5.3 : Creates an ImageDataGenerator to flow batches of grayscale images

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 60, 60, 32)	2,432
activation (Activation)	(None, 60, 60, 32)	0
max_pooling2d (MaxPooling2D)	(None, 30, 30, 32)	0
conv2d_1 (Conv2D)	(None, 28, 28, 64)	18,496
activation_1 (Activation)	(None, 28, 28, 64)	0
max_pooling2d_1 (MaxPooling2D)	(None, 14, 14, 64)	0
conv2d_2 (Conv2D)	(None, 12, 12, 64)	36,928
activation_2 (Activation)	(None, 12, 12, 64)	0
max_pooling2d_2 (MaxPooling2D)	(None, 6, 6, 64)	0
flatten (Flatten)	(None, 2304)	0
dense (Dense)	(None, 128)	295,040
dense_1 (Dense)	(None, 29)	3,741

Total params: 356,637 (1.36 MB)

Trainable params: 356,637 (1.36 MB)

Non-trainable params: 0 (0.00 B)

Fig 5.4 : Model Summary

6. SCREENSHOTS

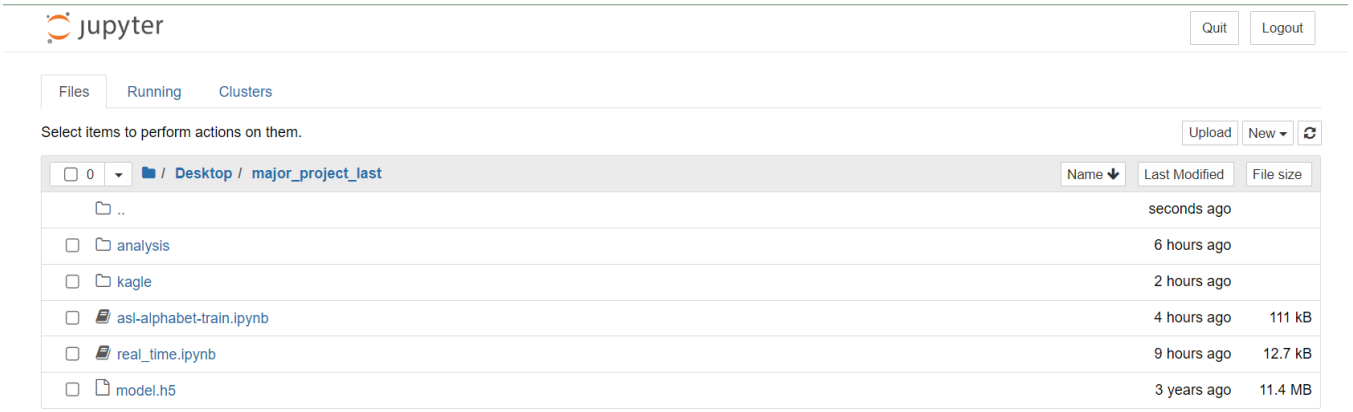


Fig 6.1 : Project Files

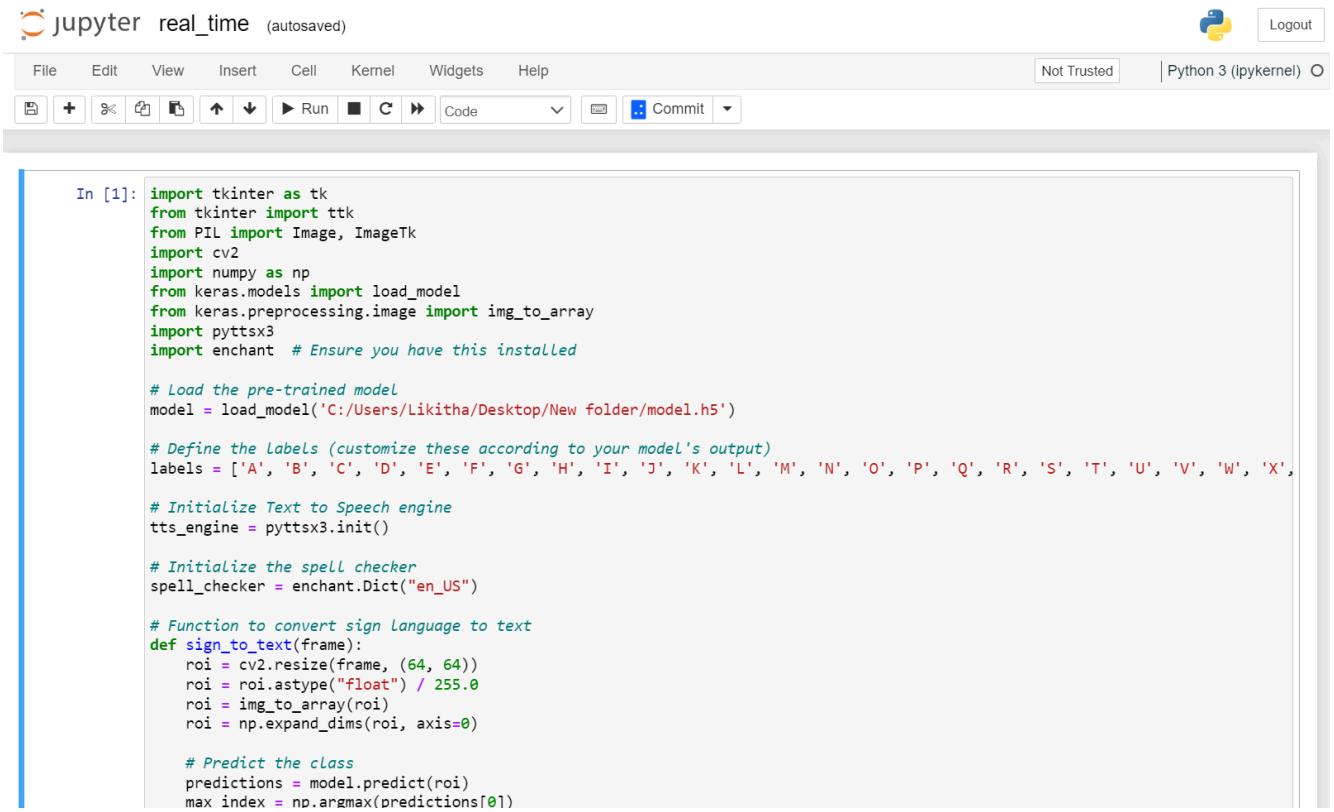
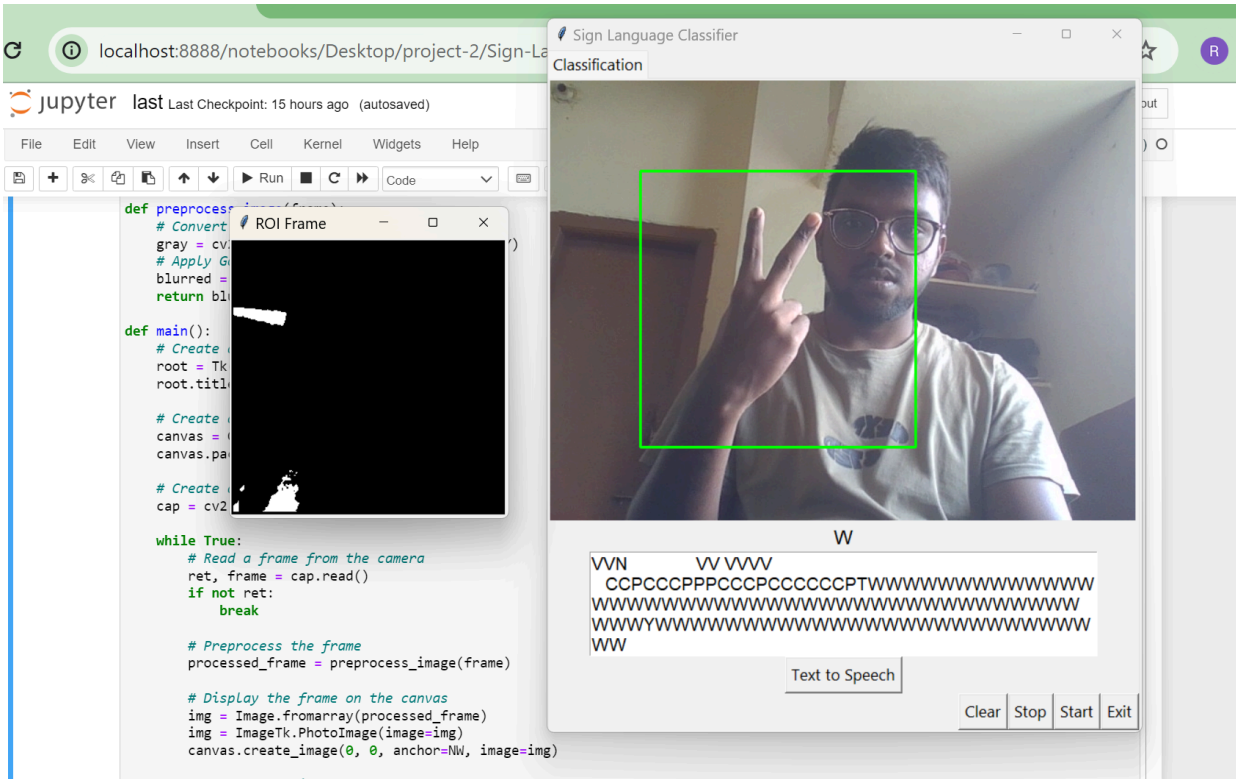


Fig 6.2 : Source Code of GUI



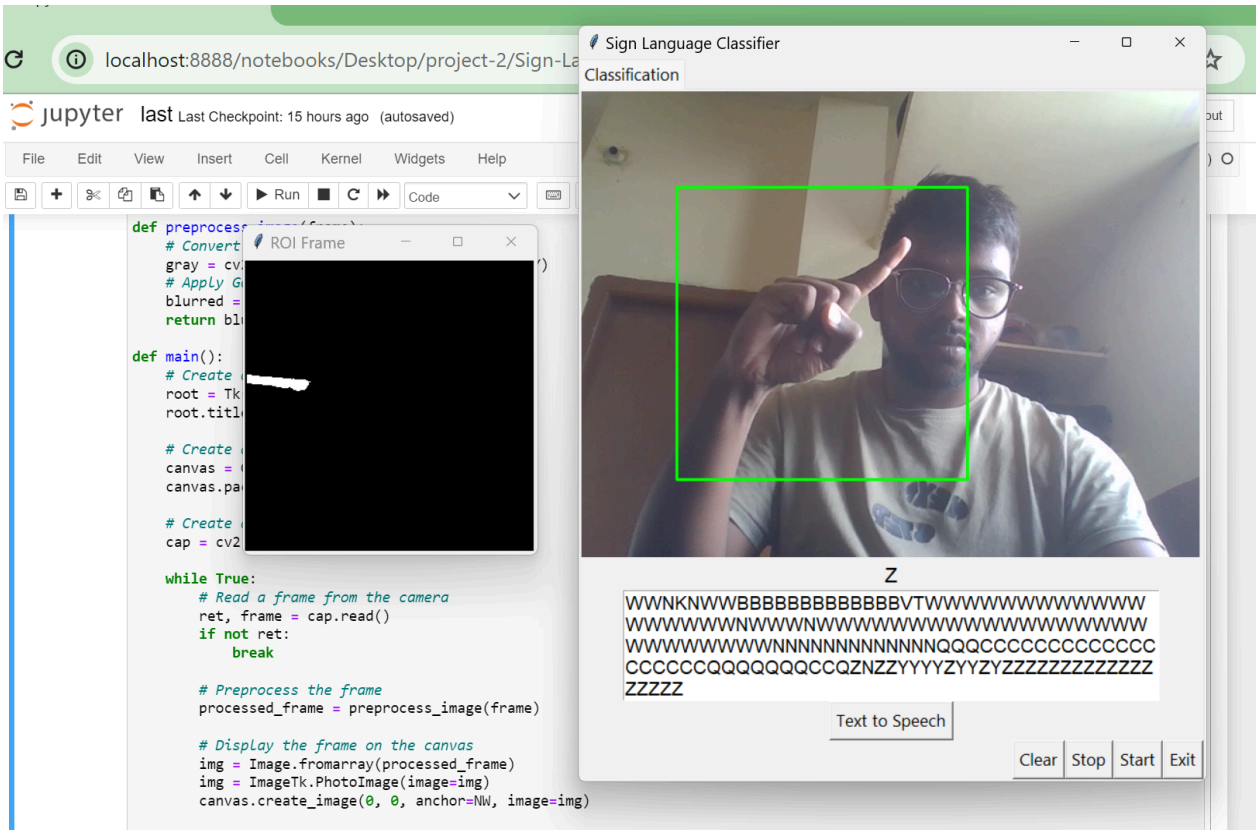


Fig 6.3 : Sign to Text and Speech

7. RESULTS AND ANALYSIS

Our proposed methodology‘s execution has been examined on the test data which was distinct from the training data set. The testing process involves 87,000 image samples of different hand signals. All these images were simultaneously updated into our proposed model to come up with accurate results. Our expected result is to obtain a text which is the translation of the sign language given as an input. Our model will anticipate all the hand gestures of the American Sign Language. To achieve an efficient result. The estimated accuracy of our proposed system is more than 95% even in a multiplex lighting environment which is considered an adequate result as of now for real-time interpretation. The entire model pipeline is developed by CNN architecture for the classification of 29 classes which 26 are for the letters A-Z and 3 classes for *SPACE*, *DELETE* and *NOTHING*. The proposed work has achieved an efficiency of 99.88%

Accuracy/Loss Training Plot

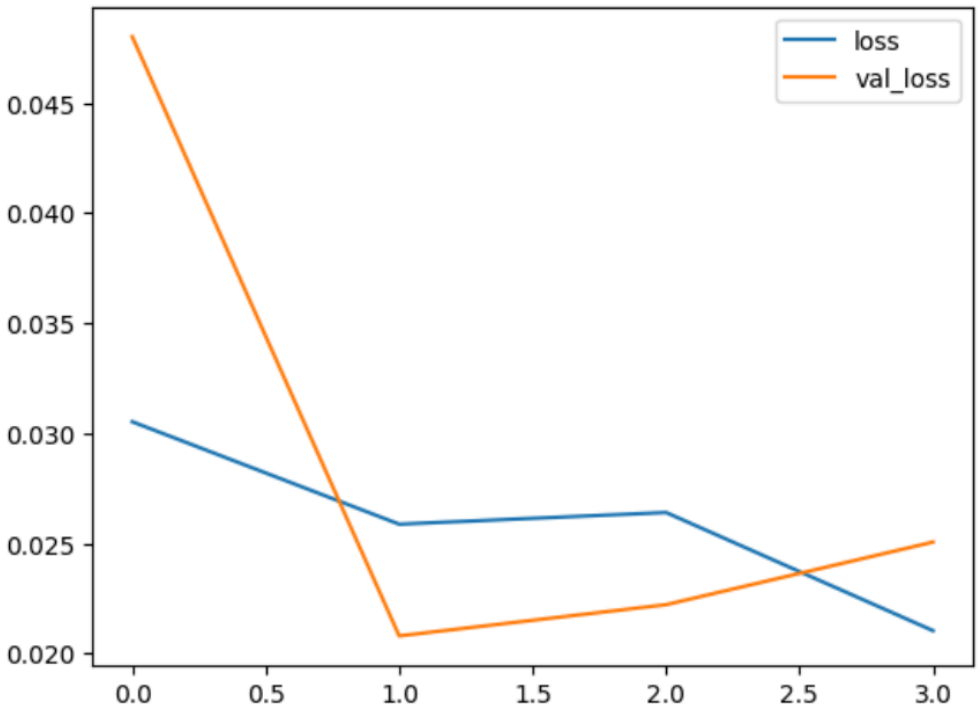


Fig 7.1 : Loss plot of the model throughout the training journey.

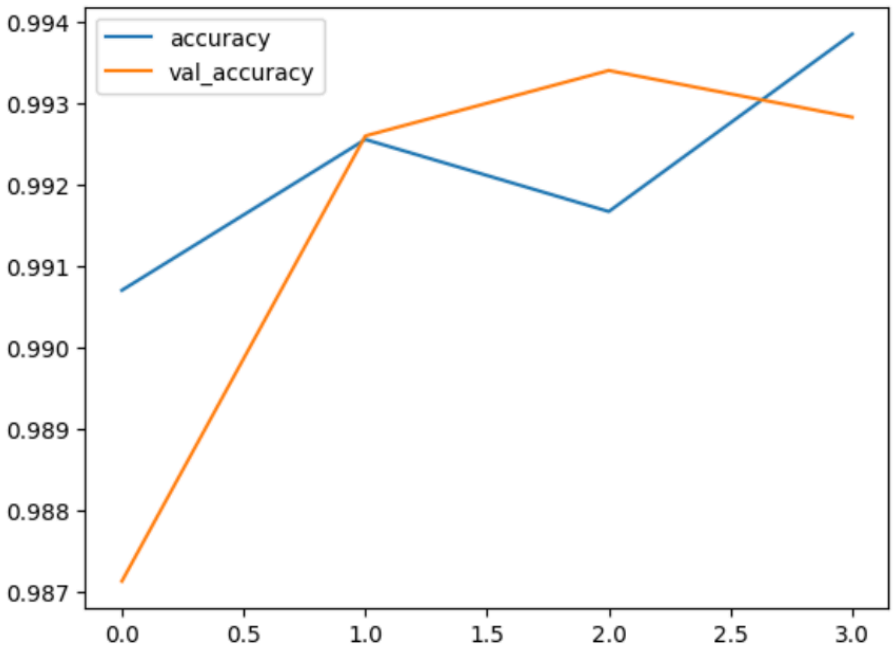


Fig 7.2 : Accuracy plot of the model throughout it's training journey

8. CONCLUSION & FUTURE ENHANCEMENT

8.1 CONCLUSION

In this project, we have presented a system for American Sign Language (ASL) recognition using a deep learning-based approach. Our system uses a convolutional neural network (CNN) trained on a dataset of hand gestures mapped to the ASL alphabet. We have demonstrated that our system achieves high accuracy in recognizing ASL gestures in real-time using live video feeds or recorded videos. Our system can be used to aid people with hearing and speech impairments in communicating with others

8.2. Future Enhancements

There is ample scope for future enhancements in our ASL recognition system. Firstly, we can expand the dataset to include a larger number of hand gestures to recognize a wider range of ASL phrases and sentences. Secondly, we can explore the use of more advanced deep learning models such as recurrent neural networks (RNNs) and attention-based models to improve accuracy. Additionally, we can investigate the use of multi-modal input sources such as audio and facial expressions to enhance the recognition capabilities of our system. Finally, we can explore the use of transfer learning techniques to adapt our system to recognize other sign languages used around the world.

9.REFERENCES

[1] Zafar Ahmed Ansari and Gaurav Harit, “Nearest Neighbour Classification of Indian Sign Language Gestures using Kinect Camera”, in Sadhana, Vol. 41, No. 2, February 2016, pp. 161- 182

[2] Lionel Pigou et al, "Sign Language Recognition using Convolutional Neural Networks", presented at the Ghent University, ELIS, Belgium

[3] Mathavan Suresh Anand, Nagarajan Mohan Kumar, Angappan Kumaresan, “An Efficient Framework for Indian SignLanguage Recognition Using Wavelet Transform” Circuits and Systems, Volume 7, pp 1874- 1883, 2016.

[4] Mandeep Kaur Ahuja, Amardeep Singh, “Hand Gesture Recognition Using PCA”, International Journal of Computer Science Engineering and Technology (IJCSET), Volume 5, Issue 7, pp. 267-27, July 2015.

[5] T. Bohra, S. Sompura, K. Parekh and P. Raut, "Real-Time Two-Way Communication System for Speech and Hearing-Impaired Using Computer Vision and Deep Learning," 2019 International Conference on Smart Systems and Inventive Technology (ICSSIT), Tirunelveli, India, 2019

[6] H. Muthu Mariappan and V. Gomathi, "Real-Time Recognition of Indian Sign Language," 2019 International Conference on Computational Intelligence in Data Science (ICCIDS), Chennai, India, 2019

[7] K. Bantupalli and Y. Xie, "American Sign Language Recognition using Deep Learning and Computer Vision," 2018 IEEE International Conference on Big Data (Big Data), Seattle, WA, USA, 2018, pp. 4896-4899, doi: 10.1109/BigData.2018.8622141.

[8] Kumar Mahesh, “Conversion of Sign Language into Text,” International Journal of Applied Engineering Research ISSN 0973- 4562 Volume 13, Number 9 (2018) pp. 7154-7161

[9] Jungong Han, Enhanced Computer Vision with Microsoft Kinect Sensor: A Review, IEEE TRANSACTIONS ON CYBERNETICS.

[10] Gunasekaran. K, Manikandan. R, International Journal of Engineering and Technology (IJET): Sign Language to Speech Translation System Using PIC Microcontroller.

[11] RiniAkmeliawatil, Melanie PO-LeenOoi et al, “Real-Time Malaysian Sign Language Translation using Color Segmentation and Neural Network. Instrumentation and Measurement Technology Conference Warsaw”, Poland.IEEE.1-6,2007.

[12] Yang quan,“Chinese Sign Language Recognition Based On Video Sequence Appearance Modeling”,IEEE. 1537-1542,2010.

[13] Wen Gao and GaolinFanga,“A Chinese sign language recognition system based on SOFM/SRN/HMM. Journal of Pattern Recognition”.2389- 2402,2004.

[14] Yun Li, Xiang Chen et al,“Sign-Component-Based Framework for Chinese Sign Language Recognition Using Accelerometer and sEMGData”,IEEE TRANSACTIONS ON BIOMEDICAL ENGINEERING, VOL. 59, NO. 10,2695-2704, 2012.

[15] Pigou, L., Dieleman, S., Kindermans, P.-J., Schrauwen, B.: Sign language recognition using convolutional neural networks. In: Workshop at the European Conference on Computer Vision 2014, pp. 572–578. Springer International Publishing (2014).

[16] Dhiman, M., 2021. Summer Research Fellowship Programme of India's Science Academies 2017. [online] AuthorCafe.

[17] T D, Sajanraj & M V, Beena. (2018). Indian Sign Language Numeral Recognition Using Region of Interest Convolutional Neural Network.

[18] Chen, J., 2021. CS231A Course Project Final Report Sign Language Recognition

[19] P.Amatya,k.Sergieva & G. Meixener," Translation of Sign Language Into Text Using Kinect for Windows v2".

[20] A. Joshi, H. Sierra & E. Arzuaga," American sign language translation using edge detection and cross-correlation".

APPENDIX

A. SAMPLE CODE

1. MODEL TRAIN FILE

```
# importing  data processing and visualisation libraries

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

%matplotlib inline

# importing  image processing libraries

import cv2

import skimage

from skimage.transform import resize

# importing  tensorflow and keras

import tensorflow as tf

from tensorflow import keras

import os

print("Packages imported...")


#Importing the dataset

batch_size = 64

imageSize = 64

target_dims = (imageSize, imageSize, 3)

num_classes = 29

train_len = 87000

train_dir = '/kaggle/input/asl-alphabet/asl_alphabet_train/asl_alphabet_train/'
```

```
def get_data(folder):
    X = np.empty((train_len, imageSize, imageSize, 3), dtype=np.float32)
    y = np.empty((train_len,), dtype=np.int_)
    cnt = 0
    for folderName in os.listdir(folder):
        if not folderName.startswith('.'):
            if folderName in ['A']:
                label = 0
            elif folderName in ['B']:
                label = 1
            elif folderName in ['C']:
                label = 2
            elif folderName in ['D']:
                label = 3
            elif folderName in ['E']:
                label = 4
            elif folderName in ['F']:
                label = 5
            elif folderName in ['G']:
                label = 6
            elif folderName in ['H']:
                label = 7
            elif folderName in ['I']:
                label = 8
            elif folderName in ['J']:
                label = 9
```

```
elif folderName in ['K']:
    label = 10
elif folderName in ['L']:
    label = 11
elif folderName in ['M']:
    label = 12
elif folderName in ['N']:
    label = 13
elif folderName in ['O']:
    label = 14
elif folderName in ['P']:
    label = 15
elif folderName in ['Q']:
    label = 16
elif folderName in ['R']:
    label = 17
elif folderName in ['S']:
    label = 18
elif folderName in ['T']:
    label = 19
elif folderName in ['U']:
    label = 20
elif folderName in ['V']:
    label = 21
elif folderName in ['W']:
    label = 22
```

```
        elif folderName in ['X']:
            label = 23
        elif folderName in ['Y']:
            label = 24
        elif folderName in ['Z']:
            label = 25
        elif folderName in ['del']:
            label = 26
        elif folderName in ['nothing']:
            label = 27
        elif folderName in ['space']:
            label = 28
        else:
            label = 29
        for image_filename in os.listdir(folder + folderName):
            img_file = cv2.imread(folder + folderName + '/' + image_filename)
            if img_file is not None:
                img_file = skimage.transform.resize(img_file, (imageSize, imageSize, 3))
                img_arr = np.asarray(img_file).reshape((-1, imageSize, imageSize, 3))

                X[cnt] = img_arr
                y[cnt] = label
                cnt += 1
    return X,y
X_train, y_train = get_data(train_dir)
print("Images successfully imported...")
```

```
#Creating  an ImageDataGenerator to flow batches of grayscale images
from tensorflow.keras.preprocessing.image import ImageDataGenerator
train_datagen = ImageDataGenerator(
  rescale=1./255,
)
train_generator = train_datagen.flow_from_directory(
  '/kaggle/input/asl-alphabet/asl_alphabet_train/asl_alphabet_train',
  target_size=(400, 400),
  batch_size=32,
  class_mode='categorical',
  color_mode='grayscale'
)

#listing the classes
class_names = list(train_generator.class_indices.keys())
print(class_names)

#checking the shape of the data
print("The shape of X_train is : ", X_train.shape)
print("The shape of y_train is : ", y_train.shape)

#Checking the shape of one image
print("The shape of one image is : ", X_train[0].shape)

#Viewing the image
plt.imshow(X_train[0])
```

```
plt.show()


#Making copies of original data

X_data = X_train

y_data = y_train

print("Copies made...")


#Train/test split Data Pre-processing

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X_data, y_data, test_size=0.3,random_state=42,stratify=y_data)


# One-Hot-Encoding the categorical data

from tensorflow.keras.utils import to_categorical

y_cat_train = to_categorical(y_train,29)

y_cat_test = to_categorical(y_test,29)


# Checking the dimensions of all the variables

print(X_train.shape)

print(y_train.shape)

print(X_test.shape)

print(y_test.shape)

print(y_cat_train.shape)

print(y_cat_test.shape)

# This is done to save CPU and RAM space while working on Kaggle Kernels. This will delete the specified data and save some space!
```

```
import gc
del X_data
del y_data
gc.collect()

#importing packages form modeling
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Activation, Dense, Flatten
print("Packages imported...")
#building models
model = Sequential()
model.add(Conv2D(32, (5, 5), input_shape=(64, 64, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(29, activation='softmax'))
model.summary()

#early stopping  and compiling
```

```
from tensorflow.keras.callbacks import EarlyStopping
early_stop = EarlyStopping(monitor='val_loss',patience=2)

#compiling
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Model fitting
model.fit(X_train, y_cat_train,
          epochs=5,
          batch_size=64,
          verbose=2,
          validation_data=(X_test, y_cat_test),
         callbacks=[early_stop])

#Metrics from model history
metrics = pd.DataFrame(model.history.history)
print("The model metrics are")
metrics

# Plotting the training loss
metrics[['loss','val_loss']].plot()
plt.show()

#Plotting the testing loss
```

```
metrics[['accuracy','val_accuracy']].plot()
plt.show()

#Model evaluation
model.evaluate(X_test,y_cat_test,verbose=0)

#Predictions
predictions=model.predict(X_test).round()
print("Predictions done...")
#saving of trained model
from keras.models import load_model
model.save('ASL.h5')
print("Model saved successfully...")

#saving of model in json formate
import json
model_json = model.to_json()
with open("ASL_DATA.json",'w') as json_file:
  json_file.write(model_json)
```

2. REAL TIME FILE

```
import tkinter as tk

from tkinter import ttk

from PIL import Image, ImageTk

import cv2

import numpy as np

from keras.models import load_model

from keras.preprocessing.image import img_to_array

import pyttsx3

import enchant  # Ensure you have this installed

# Load the pre-trained model

model = load_model('C:/Users/Likitha/Desktop/New folder/model.h5')

# Define the labels (customize these according to your model's output)

labels = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V',
'W', 'X', 'Y', 'Z', 'del', 'nothing', 'space']

# Initialize Text to Speech engine

tts_engine = pyttsx3.init()

# Initialize the spell checker

spell_checker = enchant.Dict("en_US")

# Function to convert sign language to text

def sign_to_text(frame):

   roi = cv2.resize(frame, (64, 64))

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

   roi = img_to_array(roi)

   roi = np.expand_dims(roi, axis=0)

   # Predict the class
```

```
    predictions = model.predict(roi)
    max_index = np.argmax(predictions[0])
    predicted_label = labels[max_index]

    return predicted_label
# Function to convert text to speech
def text_to_speech(text):
    tts_engine.say(text)
    tts_engine.runAndWait()
# Function to start the video feed
def start_video_feed():
    global cap, update_frame
    cap = cv2.VideoCapture(0)
    update_frame()
# Function to stop the video feed
def stop_video_feed():
    cap.release()
# Function to clear the predicted text
def clear_text():
    text_label.config(text="")
    sentence_textbox.delete('1.0', tk.END)
# Function to read text from the text box and convert it to speech
def convert_text_to_speech():
    text = sentence_textbox.get("1.0", tk.END+"-1c").strip()
    if text:
        if spell_checker.check(text):
```

```
            text_to_speech(text)
        else:
            suggested_corrections = spell_checker.suggest(text)
            if suggested_corrections:
                corrected_text = suggested_corrections[0]
            else:
                corrected_text = text
            sentence_textbox.delete("1.0", tk.END)
            sentence_textbox.insert(tk.END, corrected_text)
            text_to_speech(corrected_text)
# Initialize the main window
root = tk.Tk()
root.title("Sign Language Classifier")
# Create a notebook to hold the tabs
notebook = ttk.Notebook(root)
notebook.pack(expand=True, fill='both')
# Create the first tab
frame1 = ttk.Frame(notebook)
notebook.add(frame1, text="Classification")
# Create a frame for the video feed
frame_video = tk.Frame(frame1)
frame_video.pack()
# Create a canvas to display the video feed
canvas = tk.Canvas(frame_video, width=640, height=480)
canvas.pack()
# Create a label to display the classified text
```

```
text_label = tk.Label(frame1, text="", font=("Helvetica", 16))
text_label.pack()
# Create a text box to keep track of the sentence
sentence_textbox = tk.Text(frame1, height=5, width=50, font=("Helvetica", 14))
sentence_textbox.pack()
# Create a button to trigger text-to-speech conversion
tts_button = tk.Button(frame1, text="Text to Speech", command=convert_text_to_speech)
tts_button.pack()
# Create an exit button
exit_button = tk.Button(frame1, text="Exit", command=root.quit)
exit_button.pack(side=tk.RIGHT)
# Create a button to start the video feed
start_button = tk.Button(frame1, text="Start", command=start_video_feed)
start_button.pack(side=tk.RIGHT)
# Create a button to stop the video feed
stop_button = tk.Button(frame1, text="Stop", command=stop_video_feed)
stop_button.pack(side=tk.RIGHT)
# Create a button to clear the predicted text
clear_button = tk.Button(frame1, text="Clear", command=clear_text)
clear_button.pack(side=tk.RIGHT)
# Create the second window for the ROI frame
roi_window = tk.Toplevel(root)
roi_window.title("ROI Frame")
# Create a canvas to display the ROI frame
canvas_roi = tk.Canvas(roi_window, width=300, height=300)
canvas_roi.pack()
```

```
# Function to update the frame
def update_frame():
    ret, frame = cap.read()
    if not ret:
        return

    frame = cv2.flip(frame, 1)  # Flip the frame horizontally
    # Get the region of interest for classification (customize the ROI if needed)
    roi = frame[100:400, 100:400]
    cv2.rectangle(frame, (100, 100), (400, 400), (0, 255, 0), 2)
    # Predict the sign language
    predicted_label = sign_to_text(roi)
    text_label.config(text=predicted_label)
    # Append the predicted label to the sentence text box if it's not 'nothing'
    if predicted_label == 'space':
        sentence_textbox.insert(tk.END, ' ')
    elif predicted_label == 'del':
        current_text = sentence_textbox.get("1.0", tk.END).strip()
        current_text = current_text[:-1]
        sentence_textbox.delete("1.0", tk.END)
        sentence_textbox.insert(tk.END, current_text)
    elif predicted_label != 'nothing':
        sentence_textbox.insert(tk.END, predicted_label)
    # Convert the frame to a format that Tkinter can display
    img = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    img = Image.fromarray(img)
```

```
    img_tk = ImageTk.PhotoImage(image=img)
    # Update the canvas with the new frame
    canvas.create_image(0, 0, anchor=tk.NW, image=img_tk)
    canvas.image = img_tk
    # Process the ROI for the second window
    roi_gray = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY)
    roi_blur = cv2.GaussianBlur(roi_gray, (5, 5), 0)
    _, roi_thresh = cv2.threshold(roi_blur, 60, 255, cv2.THRESH_BINARY_INV)
    # Convert the ROI to a format that Tkinter can display
    img_roi = Image.fromarray(roi_thresh)
    img_roi_tk = ImageTk.PhotoImage(image=img_roi)
    # Update the canvas with the new ROI frame
    canvas_roi.create_image(0, 0, anchor=tk.NW, image=img_roi_tk)
    canvas_roi.image = img_roi_tk
    # Call this function again after 10 milliseconds
    root.after(10, update_frame)
# Start the Tkinter main loop
root.mainloop()
```

B. SOFTWARE AND HARDWARE REQUIREMENTS

SOFTWARE REQUIREMENTS

- Operating System : Window 10 or above , Linux
- Technology : Python
- IDE : Jupyter Notebook
- Python Version : Python 3.9 or above

HARDWARE REQUIREMENTS

- CPU : Intel i5 or Ryzen 5 or higher
- GPU : GTX 1650 or higher
- Speed : 2.6 GHz
- RAM : 8 GB

C. TECHNOLOGY USED

4.1.1 Numpy

NumPy (pronounced /ˈnʌmpaɪ/ (NUM-py) or sometimes /ˈnʌmpi/ (NUM-pee)) is a library for the Python programming language, adding support for large, multi- dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. The ancestor of NumPy, Numeric, was originally created by Jim Hugunin with contributions from several other developers. In 2005, Travis Oliphant created NumPy by incorporating features of the competing Numarray into Numeric, with extensive modifications. NumPy is open- source software and has many contributors.

Features: NumPy targets the CPython reference implementation of Python, which is a non-optimizing bytecode interpreter. Mathematical algorithms written for this version of Python often run much slower than compiled equivalents. NumPy addresses the slowness problem partly by providing multidimensional arrays and functions and operators that operate efficiently on arrays, requiring rewriting some 14 code, mostly inner loops using NumPy.

Using NumPy in Python gives functionality comparable to MATLAB since they are both interpreted,and they both allow the user to write fast programs as long as most operations work on arrays or matrices instead of scalars. In comparison, MATLAB boasts a large number of additional toolboxes, notably Simulink, whereas NumPy is intrinsically integrated with Python, a more modern and complete programming language. Moreover, complementary Python packages are available; SciPy is a library that adds more MATLAB-like functionality and Matplotlib is aplotting package that providesMATLAB-like plotting functionality. Internally, both MATLAB and NumPy rely on BLAS and LAPACK for efficient linear algebra computations.

Python bindings of the widely used computer vision library OpenCV utilize NumPy arrays to store and operate on data. Since images with multiple channels are simply represented as three-dimensional arrays, indexing, slicing or masking with other arrays are very efficient ways to access specific pixels of an image. The NumPy array as universal data structure in OpenCV for images, extracted feature points, filter kernels and many more vastly simplifies the programming workflow and debugging.

Limitations: Inserting or appending entries to an array is not as trivially possible as it is with Python's lists. The np.pad(...) routine to extend arrays actually creates new arrays of the desired shape and padding values, copies the given array into the new one and returns it. NumPy'snp.concatenate([a1,a2]) operation does not actually link the two arrays but returns a new one, filled with the entries from both given arrays in sequence. Reshaping the dimensionality of an array with np.reshape(...) is only possible as long as the number of elements in the array does not change. These circumstances originate from the fact that NumPy's arrays must be views on contiguous memory buffers. A replacement package called Blaze attempts to overcome 15 this limitation.

Algorithms that are not expressible as a vectorized operation will typically run slowly because they must be implemented in "pure Python", while vectorization may increase memory complexity of some operations from constant to linear, because temporary arrays must be created that are as large as the inputs. Runtime compilation of numerical code has been

implemented by several groups to avoid these problems; open source solutions that interoperate with NumPy include scipy.weave, numexpr and Numba. Cython and Pythran are static-compiling alternatives to these.

4.1.2 Matplotlib

Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shells, the Jupyter Notebook, web application servers, and four graphical user interface toolkits. Matplotlib tries to make easy things easy and hard things possible. You can generate plots, histograms, power spectra, bar charts, error charts, scatter plots, etc., with just a few lines of code. For examples, see the sample plots and thumbnail gallery. For simple plotting the pyplot module provides a MATLAB-like interface, particularly when combined with IPython. For the power user, you have full control of line styles, font properties, axes properties, etc, via an object oriented interface or via a set of functions familiar to MATLAB users.

4.1.3 Pandas

Pandas is a powerful open-source data manipulation and analysis library for Python. It provides easy-to-use data structures and data analysis tools that make working with structured data, such as tabular or time series data, efficient and convenient. Developed by Wes McKinney in 2008, Pandas has become a go-to library for data scientists and analysts due to its versatility and extensive functionality.

At the heart of Pandas are two primary data structures: Series and DataFrame. A Series is a one-dimensional labeled array that can hold any data type. It is similar to a column in a spreadsheet or a single column of a SQL table. A DataFrame, on the other hand, is a twodimensional labeled data structure, resembling a table or a spreadsheet, where each column can contain different data types. DataFrames are especially useful for handling structured data and performing complex data manipulations.

Pandas provides a wide range of functionalities to manipulate and transform data. It allows users to handle missing values, filter data based on conditions, sort and group data, merge and join datasets, and perform arithmetic and statistical operations. These operations can be performed with concise and expressive syntax, making data manipulation tasks more efficient.

Pandas integrates seamlessly with other popular Python libraries, such as NumPy, Matplotlib, and SciPy, enabling a comprehensive data analysis workflow. NumPy provides the underlying array processing capabilities, Matplotlib allows for data visualization, and SciPy extends the functionality with advanced statistical operations. Together with Pandas, these libraries form a powerful ecosystem for data analysis and scientific computing.

Pandas also supports reading and writing data from various file formats, including CSV, Excel, SQL databases, and more. This makes it easy to import data into Pandas from different sources, perform analysis, and export the results in a desired format. Additionally, Pandas

supports interoperability with other data analysis tools and libraries, allowing for seamless integration into existing workflows.

The Pandas community is vibrant and actively contributes to the library's development and improvement. The community provides extensive documentation, tutorials, and resources, making it easier for users to learn and utilize Pandas effectively. This collaborative environment ensures that the library remains up-to-date, reliable, and continually evolves to meet the needs of its users.

Pandas' versatility and usability have made it a popular choice not only in data science and analytics but also in a wide range of industries. It is widely used in finance, marketing, social sciences, and many other domains that require data manipulation and analysis. Its ability to handle large datasets efficiently has also made it a preferred tool for big data processing

It is built on top of another package named Numpy, which provides support for multi-dimensional arrays. As one of the most popular data wrangling packages, Pandas works well with many other data science modules inside the Python ecosystem, and is typically included in every Python distribution, from those that come with your operating system to commercial vendor distributions like Active State's Active Python.

Pandas makes it simple to do many of the time consuming, repetitive tasks associated with working with data, including:

- Data cleansing
- Data fill
- Data normalization
- Merges and joins
- Data visualization
- Statistical analysis
- Data inspection
- Loading and saving data

4.1.4 Keras

Keras is an open-source neural-network library written in Python. It is capable of running on top of TensorFlow, Microsoft Cognitive Toolkit, R, Theano, or PlaidML. Designed to enable fast experimentation with deep neural networks, it focuses on being userfriendly, modular, and extensible. It was developed as part of the research effort of project ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System), and its primary author and maintainer is François Chollet, a Google engineer. Chollet also is the author of the XCeption deep neural network model.

Features: Keras contains numerous implementations of commonly used neural- network building blocks such as layers, objectives, activation functions, optimizers, anda host of tools to make working with image and text data easier to simplify the coding necessary for writing deep neural network code.

The code is hosted on GitHub, and community support forums include the GitHub issues page, and a Slack channel. In addition to standard neural networks, Keras has support for convolutional and recurrent neural networks.

It supports other common utility layers like dropout, batch normalization, and pooling. Keras allows users to productize deep models on smartphones (iOS and Android), on the web, or on the Java Virtual Machine. It also allows use of distributed training of deep-learning models on clusters of Graphics processing units (GPU) and tensor processing units (TPU) principally in conjunction with CUDA.

Keras applications module is used to provide pre-trained model for deep neural networks. Keras models are used for prediction, feature extraction and fine tuning. This chapter explains about Keras applications in detail.

Pre-trained models Trained model consists of two parts model Architecture and model Weights. Model weights are large file so we have to download and extract the feature from ImageNet database. Some of the popular pretrained models are listedbelow,

- ResNet
- VGG16
- MobileNet
- InceptionResNetV2
- InceptionV3

4.1.5 TENSORFLOW

TensorFlow is an open-source machine learning framework developed by Google. It has gained significant popularity and has become one of the most widely used frameworks for building and deploying machine learning models. TensorFlow provides a comprehensive set of tools, libraries, and resources that enable developers to create sophisticated artificial intelligence applications.

At its core, TensorFlow is based on the concept of computational graphs. In TensorFlow, you define a computational graph that represents the flow of data and operations in a model. Nodes in the graph represent mathematical operations, while edges represent tensors, which are multi-dimensional arrays or data structures. This graph-based approach allows for efficient execution and optimization of complex computations.

TensorFlow's distributed computing capabilities are also notable. It enables training and inference on multiple machines or devices, allowing for scalable and parallel processing. This feature is particularly valuable when dealing with large datasets or complex models that require significant computational power.

Another significant advantage of TensorFlow is its extensive ecosystem. It provides a vast collection of pre-built models and components through TensorFlow Hub, making it easier for developers to leverage existing solutions and accelerate their development process. Additionally, TensorFlow offers integration with other popular libraries and tools, such as NumPy and scikit-learn, further expanding its capabilities and compatibility.

The TensorFlow Extended (TFX) ecosystem is designed specifically for end-to-end machine learning workflows. TFX provides tools for data pre-processing, model training, evaluation, and serving. It enables the deployment of machine learning pipelines at scale, making it suitable for production-level applications.

TensorFlow is not limited to traditional machine learning. It also offers support for deep learning, a subfield of machine learning that focuses on neural networks with multiple layers. Deep learning has achieved remarkable success in areas such as computer vision, natural language processing, and speech recognition. TensorFlow's deep learning capabilities, coupled with its computational efficiency, make it a popular choice for building and training deep neural networks.

The TensorFlow community is vibrant and active. It has a large and growing user base that contributes to the development and improvement of the framework. The community provides extensive documentation, tutorials, and resources, making it easier for beginners to get started with TensorFlow and for experienced developers to explore advanced techniques.

4.1.6 OpenCV

OpenCV (Open Source Computer Vision Library) is a library of programming functions mainly for real-time computer vision. Originally developed by Intel, it was later supported by Willow Garage, then Itseez (which was later acquired by Inte). The library is cross-platform and licensed as free and open-source software under Apache License 2. Starting in 2011, OpenCV features GPU acceleration for real-time operations.

OpenCV is a general-purpose programming language started by Guido van Rossum that became very popular very quickly, mainly because of its simplicity and code readability. It enables the programmer to express ideas in fewer lines of code without reducing readability. Compared to languages like C/C++ Python is slower. That said, Python can be easily extended with C/C++, which allows us to write computationally intensive code in C/C++ and create Python wrappers that can be used as Python modules. This gives us two advantages: first, the code is as fast as the original C/C++ code (since it is the actual C++ code working in background) and second, it easier to code in Python than C/C++. OpenCV-Python is a Python wrapper for the original OpenCV C++ implementation.

OpenCV-Python makes use of Numpy, which is a highl optimized library for numerical operations 10 with a MATLAB-style syntax. All the OpenCV array structures are converted to and from Numpy arrays. This also makes it easier to integrate with other libraries that us Numpy such as SciPy and Matplotlib

OpenCV has a modular structure, which means that the package includes several shared or static libraries. The following modules are available:

- **Core functionality** (**core**) - a compact module defining basic data structures, including the dense multi-dimensional array Mat and basic functions used by all other modules.

- **Image Processing** (**imgproc**) - an image processing module that includes linear and non-linear image filtering, geometrical image transformations (resize, affine and perspective warping, generic table-based remapping), color space conversion, histograms, and so on.
- **Video Analysis** (**video**) - a video analysis module that includes motion estimation, background subtraction, and object tracking algorithms.
- **Camera Calibration and 3D Reconstruction** (**calib3d**) - basic multiple-view geometry algorithms, single and stereo camera calibration, object pose estimation, stereo correspondence algorithms, and elements of 3D reconstruction.
- **2D Features Framework** (**features2d**) - salient feature detectors, descriptors, and descriptor matchers.
- **Object Detection** (**objdetect**) - detection of objects and instances of the predefined classes (for example, faces, eyes, mugs, people, cars, and so on).
- **High-level GUI** (**highgui**) - an easy-to-use interface to simple UI capabilities.
- **Video I/O** (**videoio**) - an easy-to-use interface to video capturing and video codecs.
- some other helper modules, such as FLANN and Google test wrappers, Python bindings, and others.

4.1.7 Tkinter

Tkinter is a Python library that serves as the standard GUI (Graphical User Interface) toolkit for Python. It allows developers to create desktop applications with graphical interfaces in a straightforward and efficient manner. Tkinter is included with most Python installations, making it readily available for development without the need for additional installations.

Tkinter provides a simple and intuitive interface for creating GUI applications, making it suitable for both beginners and experienced developers. Tkinter applications are platform-independent and can run seamlessly on various operating systems such as Windows, macOS, and Linux, ensuring broad compatibility.

Tkinter offers a wide range of built-in widgets (GUI elements) such as buttons, labels, entry fields, checkboxes, radio buttons, menus, and more, enabling developers to design versatile and interactive interfaces. Developers can customize the appearance and behavior of widgets using various configuration options, including colors, fonts, sizes, alignments, and event bindings, allowing for tailored user experiences.

Tkinter follows an event-driven programming paradigm, where actions or events (e.g., button clicks, mouse movements) trigger corresponding event handlers or callback functions, facilitating responsive and interactive applications.

Tkinter seamlessly integrates with Python's syntax and features, enabling developers to leverage Python's robust capabilities for data processing, computation, and logic within GUI applications. Tkinter benefits from a large and active community of developers who contribute tutorials, documentation, and third-party resources, fostering learning and collaboration among users.

Tkinter serves as a powerful and accessible tool for developing desktop GUI applications in Python. Its simplicity, versatility, and cross-platform compatibility make it an excellent choice for projects ranging from simple utilities to complex software applications. With Tkinter, developers can create intuitive and visually appealing user interfaces to enhance the usability and functionality of their Python applications.

4.1.8 Enchant

Enchant is a powerful and versatile spellchecking library for Python that provides developers with the tools to incorporate spellchecking functionality into their applications seamlessly. It offers support for multiple languages and dictionaries, making it suitable for a wide range of projects requiring text validation and correction capabilities.

Enchant is designed to work across various platforms, including Windows, macOS, and Linux, ensuring consistent spellchecking functionality across different operating systems.

Enchant supports multiple languages and dictionaries, allowing developers to perform spellchecking in different languages based on their application requirements. Enchant employs efficient algorithms for spellchecking, enabling fast and accurate identification of misspelled words and suggestions for corrections.

Enchant provides developers with the flexibility to extend and customize the spellchecking functionality to suit their specific needs. Users can add custom dictionaries, define personal word lists, and modify settings to enhance spellchecking accuracy and efficiency.

Enchant seamlessly integrates with the Python programming language, making it easy for developers to incorporate spellchecking capabilities into their Python applications using familiar syntax and conventions.

Enchant offers a user-friendly interface for interacting with spellchecking functionality, providing intuitive methods for checking spelling, obtaining suggestions, and performing corrections within the application.Enchant is an open-source project with an active community of developers and contributors. Users can benefit from community support, documentation, and resources to troubleshoot issues, learn new features, and collaborate with other users.

Enchant is a valuable tool for developers seeking to enhance the quality and professionalism of their text-based applications by incorporating spellchecking capabilities. Its cross-platform compatibility, support for multiple languages, and customizable features make it an indispensable library for projects requiring robust text validation and correction functionality. With Enchant, developers can ensure that their applications deliver accurate and error-free content, enhancing the user experience and overall quality of the software.

4.1.9 pyttsx3

pyttsx3 is a Python library that provides text-to-speech (TTS) functionality, allowing developers to convert text into spoken audio. It is a powerful tool for creating applications that interact with users through speech, such as virtual assistants, accessibility tools, and educational software. With pyttsx3, developers can easily incorporate speech synthesis

capabilities into their Python projects, enabling them to deliver information and communicate with users in a natural and accessible manner.

pyttsx3 is compatible with various platforms, including Windows, macOS, and Linux, ensuring consistent speech synthesis functionality across different operating systems.

pyttsx3 supports multiple speech engines, including SAPI5 on Windows, NSSpeechSynthesizer on macOS, and espeak on Linux, providing users with options to choose the preferred speech synthesis engine based on their requirements.

pyttsx3 offers customization options for speech output, allowing developers to adjust parameters such as speech rate, volume, and voice characteristics to tailor the speech synthesis to their application needs.

pyttsx3 supports text formatting features such as emphasis, pitch, and pronunciation adjustments, enabling developers to control the intonation and expression of the synthesized speech.

pyttsx3 seamlessly integrates with the Python programming language, making it easy for developers to incorporate speech synthesis functionality into their Python applications using familiar syntax and conventions.

pyttsx3 supports asynchronous operation, allowing developers to generate speech output in the background while the main application continues executing other tasks, enhancing responsiveness and user experience.

pyttsx3 is an open-source project with an active community of developers and contributors. Users can benefit from community support, documentation, and resources to troubleshoot issues, learn new features, and collaborate with other users.

pyttsx3 is a versatile and user-friendly library for adding text-to-speech capabilities to Python applications. Its cross-platform compatibility, support for multiple speech engines, and customizable features make it an indispensable tool for projects requiring speech synthesis functionality. With pyttsx3, developers can create engaging and accessible applications that communicate with users through spoken audio, enhancing usability and accessibility for a wide range of users.

4.1.10 Python Imaging Library (PIL)

The Python Imaging Library, commonly known as PIL, is a powerful and versatile library for image processing tasks in Python. It provides developers with a wide range of functionalities for manipulating and enhancing digital images, making it a valuable tool for various applications such as computer vision, image analysis, and graphical user interfaces.

PIL allows developers to perform a variety of image manipulation operations, including resizing, cropping, rotating, flipping, and transforming images. These operations enable developers to modify images to suit their specific requirements.

PIL provides tools for enhancing the quality and appearance of images, such as adjusting brightness, contrast, saturation, and sharpness. These enhancements can improve the visual appeal and clarity of images for better analysis and presentation.

PIL offers a wide range of image filtering techniques, including blurring, sharpening, edge detection, and noise reduction. These filters can help remove imperfections, highlight important features, and enhance image details for better analysis and interpretation.

PIL allows developers to draw shapes, lines, text, and other graphical elements directly onto images. This functionality is useful for annotating images, adding labels, and creating visualizations for analysis and presentation purposes.

PIL supports a variety of image file formats, including JPEG, PNG, GIF, BMP, TIFF, and more. This extensive file format support enables developers to read, write, and manipulate images in different formats seamlessly.

PIL seamlessly integrates with the Python programming language, making it easy for developers to incorporate image processing capabilities into their Python applications using familiar syntax and conventions.

PIL is an open-source project with an active community of developers and contributors. Users can benefit from community support, documentation, and resources to troubleshoot issues, learn new features, and collaborate with other users.

The Python Imaging Library (PIL) is a versatile and user-friendly library for image processing tasks in Python. Its comprehensive set of features, including image manipulation, enhancement, filtering, drawing, and file format support, makes it an invaluable tool for developers working with digital images. With PIL, developers can easily perform a wide range of image processing operations to analyze, manipulate, and enhance images for various applications.

RESOURCES

Scan the QR Code for References Pdf