SIGN LANGUAGE TRANSLATOR

Project Report submitted to

Guru Jambheshwar University of Science and Technology, Hisar

for the partial award of the degree

Of

Master of Computer Application

by

Chandrakanta (210010120012)
Khushboo Rani (210010120018)
MCA Final Year
GJUS&T, Hisar

Dr. Amandeep Noliya(Asst. Professor)
Dr. Deepak Nandal(Asst. Professor)
Department of CSE
GJUS&T, Hisar

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

GURU JAMBHESHWAR UNIVERSITY OF SCIENCE AND TECHNOLOGY, HISAR

JUNE 2023

DECLARATION

We, Chandrakanta, 210010120012 and Khushboo Rani, 210010120018 certify that the work contained in this project report is original and has been carried by us under the guidance of Dr. Amandeep Noliya and Dr. Deepak Nandal. This work has not been submitted to any other institute for the award of any degree or diploma and we have followed the ethical practices and other guidelines provided by the Department of Computer Science and Engineering in preparing the report. Whenever we have used materials (data, theoretical analysis, figures, and text) from other sources, we have given due credit to them by citing them in the text of the report and giving their details in the references.

Signature

Chandrakanta
210010120012
Department of CSE
GJUS&T, Hisar

Signature

Khushboo Rani
210010120018
Department of CSE
GJUS&T, Hisar

Signature

Dr. Amandeep Noliya
Designation: - **Assistant Professor**
Department of CSE
GJUS&T, Hisar

Signature

Dr. Deepak Nandal
Designation: - **Assistant Professor**
Department of CSE
GJUS&T, Hisar

DECLARATION

We, Chandrakanta, 210010120012 and Khushboo Rani, 210010120018 certify that the work contained in this project report is original and has been carried by us under the guidance of Dr. Amandeep Noliya and Dr. Deepak Nandal. This work has not been submitted to any other institute for the award of any degree or diploma and we have followed the ethical practices and other guidelines provided by the Department of Computer Science and Engineering in preparing the report. Whenever we have used materials (data, theoretical analysis, figures, and text) from other sources, we have given due credit to them by citing them in the text of the report and giving their details in the references.

Signature Saini 28/6/23

Chandrakanta
210010120012
Department of CSE
GJUS&T, Hisar

Signature Khushboo 28/06/23

Khushboo Rani
210010120018
Department of CSE
GJUS&T, Hisar

Signature

Dr. Amandeep Noliya
Designation: - **Assistant Professor**
Department of CSE
GJUS&T, Hisar

Signature

Dr. Deepak Nandal
Designation: - **Assistant Professor**
Department of CSE
GJUS&T, Hisar

CERTIFICATE

This is certified that Chandrakanta (210010120012), Khushboo Rani (210010120018) has worked under my supervision to prepare his project on "SIGN LANGUAGE TRANSLATOR". They have worked on their project through the semester from MARCH 2023 to JUNE 2023.

I wish him success in life.

Dr. Amandeep Noliya
Assistant Professor
Department of CSE
GJUS&T, Hisar

Dr. Deepak Nandal
Assistant Professor
Department of CSE
GJUS&T, Hisar

CERTIFICATE

This is certified that Chandrakanta (210010120012), Khushboo Rani (210010120018) has worked under my supervision to prepare his project on "SIGN LANGUAGE TRANSLATOR". They have worked on their project through the semester from MARCH 2023 to JUNE 2023.

I wish him success in life.

Dr. Amandeep Noliya
Assistant Professor
Department of CSE
GJUS&T, Hisar

Dr. Deepak Nandal
Assistant Professor
Department of CSE
GJUS&T, Hisar

PLAGIARISM CERTIFICATE

This is certify that Chandrakanta (210010120012), Khushboo Rani (210010120018) are students of MCA (CSE), Department of computer Science & Engineering, Guru Jambheshwar University of Science & Technology, Hisar has completed the project entitle "SIGN LANGUAGE TRANSLATOR".

This complete project report has been checked by Turnitin Software and the similarity index is 9% i.e. the accepted norms of university. The project report may be considered for the award of the degree.

Signature:
Chandrakanta(210010120012)
MCA Final Year
Department of CSE, GJUS&T, Hisar

Signature:
Khushboo Rani (210010120018)
MCA Final Year
Department of CSE, GJUS&T, Hisar

PLAGIARISM CERTIFICATE

This is certify that Chandrakanta (210010120012), Khushboo Rani (210010120018) are students of MCA (CSE), Department of computer Science & Engineering, Guru Jambheshwar University of Science & Technology, Hisar has completed the project entitle "SIGN LANGUAGE TRANSLATOR".

This complete project report has been checked by Turnitin Software and the similarity index is 9% i.e. the accepted norms of university. The project report may be considered for the award of the degree.

Signature:
Chandrakanta(210010120012)
MCA Final Year
Department of CSE, GJUS&T, Hisar

Signature:
Khushboo Rani (210010120018)
MCA Final Year
Department of CSE, GJUS&T, Hisar

ACKNOWLEDGEMENT

An assignment puts to the litmus test an individual's knowledge, credibility, and experience and thus sole efforts of an individual are not sufficient to accomplish the desire. Successful Completion of the project involves interest and efforts of many people. So, this becomes Obligatory on my part to record my thanks to all of them.

Therefore, in this way would like to thank Dr. Amandeep Noliya & Dr. Deepak Nandal (Department of CSE). The project supervisor, who had allowed us to work and listened to our progress report timely, He had discussed about the project and suggested us to work in enhanced manner.

At last, but not least, we express our heartiest gratitude to all our friends and Supervisor and all in all almighty for providing us a favourable environment and support.

Signature:
Chandrakanta (210010120012)
MCA Final Year
Department of CSE, GJUS&T, Hisar

Signature:
Khushboo Rani (210010120018)
MCA Final Year
Department of CSE, GJUS&T, Hisar

ACKNOWLEDGEMENT

An assignment puts to the litmus test an individual's knowledge, credibility, and experience and thus sole efforts of an individual are not sufficient to accomplish the desire. Successful Completion of the project involves interest and efforts of many people. So, this becomes Obligatory on my part to record my thanks to all of them.

Therefore, in this way would like to thank Dr. Amandeep Noliya & Dr. Deepak Nandal (Department of CSE). The project supervisor, who had allowed us to work and listened to our progress report timely, He had discussed about the project and suggested us to work in enhanced manner.

At last, but not least, we express our heartiest gratitude to all our friends and Supervisor and all in all almighty for providing us a favourable environment and support.

Signature: 28/6/23

Chandrakanta (210010120012)
MCA Final Year
Department of CSE, GJUS&T, Hisar

Signature: 28/06/23

Khushboo Rani (210010120018)
MCA Final Year
Department of CSE, GJUS&T, Hisar

LIST OF FIGURES

Figure No.	Name	Page No.
Figure 1.1	Sign language symbols	7
Figure 1.2	Architecture	11
Figure 3.1	VS Code Environment	23
Figure 4.1	Output of collectiondata.py	32
Figure 4.2	function.py file	35
Figure 4.3	Output of data.py	39
Figure 4.4	Output of trainmodel.py	42
Figure 4.5	Output of app.py	48

TABLE OF CONTENTS

Chapter No.	Title	Page No.
	Declaration	i
	Certificate	ii
	Plagiarism Certificate	iii
	Acknowledgement	iv
	List of figure	v
	Table of Content	vi-vii
	Abstract	viii
1. CHAPTER :	**INTRODUCTION**	**1-16**
	1.1 Language translator	1-7
	1.1.1 Language translator types	1
	1.2 Sign Language	7
	1.3 History	8-10
	1.4 Sign language translator	10-16
	1.4.1 Sign language translator architecture	11
	1.4.2 Need of Sign language translator	12
	1.4.3 Challenges of Sign language translator	13
	1.4.4 Working of Sign language translator	14
	1.4.5 Application of language translator	15-16
2. CHAPTER :	**LITERATURE REVIEW**	**17-20**
	2.1 Problem Statement	20
3. CHAPTER :	**METHODOLOGY**	**21-25**
	3.1Working of sign language translator	21
	3.1.1 Convolutional layer	21
	3.2 Technologies used	22
	3.2.1 Visual studio code	22
	3.2.2 Open cv	23

3.2.3 Media pipe 24

3.2.4 Tensor flow 24

3.2.5 Keras 25

4. CHAPTER : OBJECTIVE 26-49

4.1 Data Collection 26-33

4.2 Functioning 33-36

4.3 Data.py 36-40

4.4 Training Model 40-44

4.5 Final output 44-49

5. CHAPTER: CONCLUSION 50

5.1 Future Scope 50

REFERENCES 51-53

Plagiarism certificate 54-55

ABSTRACT

The Sign Language Translator is an innovative technology designed to facilitate seamless communication between deaf and hearing individuals. Sign language serves as the primary means of communication for many deaf individuals, but it presents a significant barrier for those who do not understand or use sign language. This project aims to bridge the communication gap by developing a real-time sign language translation system that can accurately convert sign language gestures into spoken or written language, and vice versa.

The Sign Language Translator utilizes a combination of computer vision, machine learning, and natural language processing techniques to interpret sign language gestures. The system is equipped with a camera or sensor that captures the user's sign language movements. These visual inputs are processed and analyzed by the underlying algorithms, which recognize the specific signs and gestures being performed.

To achieve accurate translation, the system employs a deep learning model trained on a vast dataset of sign language gestures. The model recognizes and classifies individual signs, as well as considers the contextual meaning of the gestures to ensure accurate translations. The translated output can be presented in multiple formats, including spoken language through text-to-speech synthesis or written language via a display interface.

The Sign Language Translator has the potential to revolutionize communication for deaf individuals, empowering them to interact more effectively with the hearing community. It can be implemented in various settings, such as educational institutions, healthcare facilities, public services, and everyday conversations. By breaking down the communication barriers, the Sing Language Translator promotes inclusivity and equal access to information for deaf individuals, fostering a more inclusive society.

CHAPTER-1
INTRODUCTION

1.1 Language translator:-

A language translator is a software tool that is designed to convert text o from one language into other language.[1] It is used to help people communicate across different languages, whether in written or spoken form. Language translators work by analyzing the grammar, syntax, and vocabulary of the input text or speech and then generating an output text or speech in the desired language. There are many different types of language translators available, ranging from simple online tools to complex machine learning algorithms. Some popular examples include Google Translate, Microsoft Translator, and DeepL. Language translators can be found in various forms, such as web-based translation services, mobile applications, desktop software, or integrated into specific devices like Smartphone or voice assistants. They support multiple language pairs, allowing users to translate between different languages depending on their requirement. It is significant to note that while language translators have advanced significantly in recent years, they may still have limitations in accurately capturing nuances, idiomatic expressions, and cultural context. Human translation is generally considered more reliable for complex or sensitive content.[1]

1.1.1 Language translator types:

1.1.1(1) Mobile translator

These are translation apps that can be installed on mobile devices and used to translate text, speech, or images on the go A mobile translator is a type of language translation tool or application these are installed and used on mobile devices like smart phones or tablets. It enables users to translate text or speech from one language to another directly on their mobile devices, making it convenient for on-the-go translation needs. Mobile translators can provide various features and functionalities, depending on the specific app or platform.

Some common features of mobile translator apps include:

- **Text Translation:** Mobile translators allow users to type or paste text in one language and receive an instant translation in another language. This feature is useful for translating written content such as messages, emails, or documents.
- **Speech Translation:** Mobile translators can also provide speech-to-text and text-to-speech translation capabilities. Users can speak or dictate in one language, and the app will convert it into text and then translate it into another language. This feature is handy for real-time communication or when faced with unfamiliar written text.
- **Camera Translation:** Many mobile translator apps offer camera translation, where users can capture an image of written text, such as signs, menus, or documents, and the app will recognize the text and provide a translation. This feature is beneficial when encountering foreign language text in the physical environment.
- **Offline Translation:** Some mobile translators offer offline translation functionality, allowing users to download language packs and use the app without an internet connection. This is particularly useful when traveling to areas with limited or no internet access.
- **Language Learning Resources:** Certain mobile translator apps may include additional language learning features, such as vocabulary lists, phrasebooks, or pronunciation guides, to help users improve their language skills.[2]

Mobile translators are available for various platforms, including iOS (Apple) and Android devices. They often support multiple language pairs and offer a user-friendly interface for easy navigation and interaction.

1.1.1(2) Rule-based translators

These translators work by following a set of pre-defined rules and linguistic patterns to translate text from one language to another. A rule-based translator is a type of language translation system that follows a set of predefined linguistic rules

to perform translations. It relies on linguistic rules and patterns to analyze the structure and meaning of sentences in the source language and generate corresponding translations in the target language. Here's a brief overview of how rule-based translation works.

- **Linguistic Rules**: Rule-based translation systems use a set of linguistic rules that describe the grammar, syntax, and vocabulary of both the source and target languages. These rules are manually created and maintained by language experts or linguists.
- **Analysis:** The rule-based translator analyzes the input text or speech in the source language, breaking it down into grammatical units such as words, phrases, or sentences. It applies linguistic rules to analyze the structure, grammar, and semantics of the text.
- **Rule Application:** Based on the linguistic rules, the translator applies transformation rules that define how the source language elements should be converted into the target language. These rules can include word-for-word translations, syntactic transformations, or semantic adjustments.
- **Generation:** Once the analysis and rule application are complete, the rule-based translator generates the translated output in the target language. The output is constructed using the transformed linguistic elements based on the predefined rules.

Rule-based translation systems are typically designed and customized for specific language pairs and domains. They require a comprehensive set of linguistic rules, extensive lexicons or dictionaries, and syntactic and semantic knowledge. These systems can handle complex language structures and maintain grammatical accuracy.

1.1.1(3) Statistical machine translators

These translators use statistical models and algorithms to translate text based on the patterns and structures found in large data sets of bilingual texts. SMT (statistical machine translator) is an approach to language translation that relies on statistical models and algorithms. It uses large bilingual corpora (collections of

translated texts) to automatically learn the probabilistic relationships between words, phrases, and sentences in different languages. Statistical machine translators are built based on these learned patterns to generate translations. Here's a brief overview of how statistical machine translation works. [2]

- **Training Phase:** In the training phase, a statistical machine translator is exposed to a vast amount of parallel bilingual data. This data consists of source language sentences paired with their corresponding translations in the target language. The translator analyzes this data to extract statistical patterns and relationships between words, phrases, and sentence structures.
- **Alignment and Probability Estimation:** During training, the statistical machine translator aligns the source and target sentences to identify correspondences between words and phrases. It builds probabilistic models that estimate the likelihood of a translate a specific word or phrase from the source language into the target language.
- **Translation Generation:** When a new input sentence is given for translation, the statistical machine translator uses the learned probabilities to generate the most likely translation. It calculates the probability distribution over possible translations and selects the translation with the highest probability based on the learned models.
- **Post-Processing:** After the initial translation is generated, Refinement steps may be implemented to improve the fluency and naturalness of the translation. These steps can include adjusting word order, resolving grammatical errors, or adapting the translation to specific stylistic or domain requirements.
- **Neural machine translators:** These translators are an advanced form of statistical machine translation that uses artificial neural networks to improve translation accuracy and quality. Neural machine translation (NMT) is an advanced approach to language translation that uses artificial neural networks, specifically recurrent neural networks (RNNs) or transformer models, to generate translations. NMT has gained significant attention and has become the dominant method in machine translation due to its ability to capture complex linguistic patterns and produce fluent and context-aware translations.[3]

Here's a brief overview of how neural machine translation works:

- **Training Phase:** In the training phase, a neural machine translator is teached on a big parallel corpus, which consists of aligned sentences in the source and target languages. The neural network is designed to learn the relationships between the input sentences and their corresponding translations.
- **Encoder-Decoder Architecture:** Neural machine translators typically employ. An encoder-decoder architecture is utilized, where the encoder analyzes the source sentence and converts it into a predetermined, unchanging representation known as a "context vector" or "thought vector." This vector encapsulates the essence and context of the source sentence. The decoder then takes this context vector and generates the translation by predicting the target words one by one.
- **Neural Network Learning:** During training, the neural network learns to map the source sentences to their corresponding translations. It learns the patterns and dependencies between words and phrases in both languages by adjusting the weights of the neural network parameters through a process called back-propagation. This process minimizes the Distinction between the predicted translations and the actual translations in the training data.
- **Translation Generation:** Once the neural machine translator is trained, it can be used to translate new input sentences. The input sentence is encoded by the encoder, and the decoder generates the translation based on the learned patterns and context from the training phase. The decoder generates the translation word by word, considering the previous words and the context vector.

Advantages of Neural Machine Translation:

- Neural machine translators have demonstrated superior performance in terms of translation quality, fluency, and capturing contextual information compared to earlier approaches.
- They can handle long sentences and complex language structures more effectively.
- Neural networks have the capability to learn from large amounts of data and can benefit from continuous improvement as more data becomes available.

- NMT models can be trained for various language pairs and domains, making them flexible and adaptable.

1.1.1(4) Sign language translator:

A sign language to text language translator is a type of technology that converts sign language gestures into written or spoken language. This can be done using a variety of methods, such as through computer vision and machine learning algorithms or through sensors that track the movement of the hands and body[3] sign language translator is a system or device designed to facilitate Interaction between deaf or Hearing impaired persons who use gesture based language and those who doesn't understand gestures . Sign language translators aim to bridge the communication gap by converting sign language into spoken or written language, or vice versa[25].

Types of sign language translators:

- **Human Translators:** Professional Gesture-based language interpreters are trained individuals who act as intermediaries between Hearing impaired and non-Hearing impaired. They listen to spoken language and interpret it into sign language for the deaf individual, and vice versa. Human interpreters are highly skilled in both sign and the target verbal language and can convey nuances and cultural aspects of communication.
- **Computer-Based Translators:** Computer-based sign language translators utilize technology to convert spoken or written language into Gesture-based language or vice versa. These systems commonly utilize a blend of computer vision, machine learning, and natural language processing methods to achieve their functionality. They use cameras or sensors to capture hand sign gestures and convert them into text or speech, or analyze Verbal language and generate sign language animations or video representations.
- **Wearable Devices:** There are wearable devices being developed that aim to make communication between deaf users and normal users. These devices may include sensors or cameras that capture sign language gestures, which are then processed and converted into spoken or written language output..

Mobile applications are being developed to assist in sign language translation. These apps may include sign language dictionaries or phrasebooks that provide translations of common words, phrases, or sentences between Gesture-based language and written language. Some apps also use video or animation to demonstrate sign language gestures.

1.2 Sign Language

Sign language is visualized language that is used by people who are deaf or to interact with others[24]. It uses a combination of hand gestures, facial expressions, and body movements to deliver meaning. Sign languages vary from country to country and region to region, and there are many different hand gesture languages in use around the world[26]. Sign languages are not simply a visual representation of verbal language, but rather they have their own grammar, syntax, and vocabulary[5]

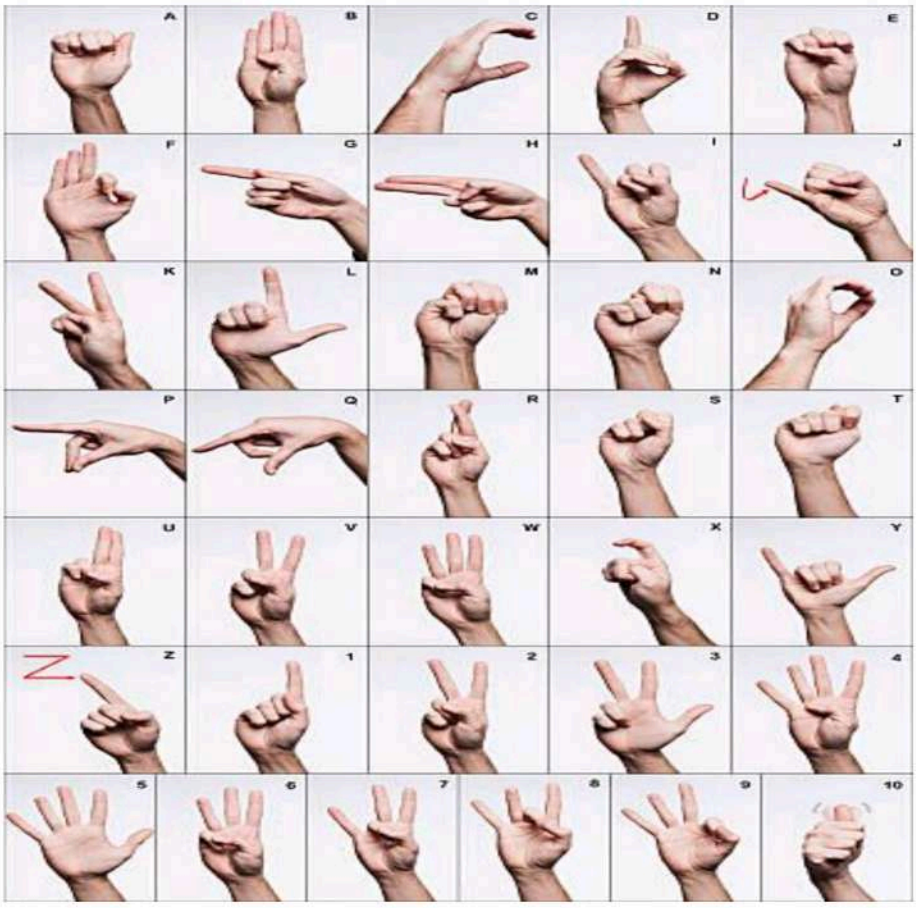

Figure 1.1 Sign language symbols[6]

Learning sign language can be beneficial for both deaf and hearing individuals, as it can promote better communication, understanding, and inclusion.

1.3 HISTORY

The Historical background of sign language translation is Intimately connected to the development and recognition of sign languages themselves.

Here's a brief overview:

- **Early Efforts:** The need for sign language translation emerged as societies began to recognize the importance of including deaf individuals in education, employment, and public life. In the 18th and 19th centuries, there were isolated instances of individuals acting as sign language interpreters, although formal systems for sign language translation had not yet been established. Sign languages have existed for centuries, with evidence of manual communication systems dating back to ancient civilizations. In many societies, deaf individuals developed their own natural sign languages within their communities, serving as a means of communication and cultural expression.
- **Manual Alphabet Systems:** One early form of hand gesture language translation involved the use of manual alphabet systems. These systems, like as the British Two-Handed Alphabet and the French Sign Language Manual Alphabet, relied on finger spelling to represent written languages. They enabled deaf individuals to communicate with the hearing population in written form but did not capture the full complexity of sign languages.
- **Formalization:** Formalization & recognition of hand gesture languages began in the 18th century. In 1755, the first comprehensive book on sign language, "Essay on the Education of the Deaf," was written a French educator. He developed a system called Old French Sign Language (LSF) and founded the first public school for the hard to hearing people in Paris.
- **Regional Variation:** As sign languages spread and evolved, regional variations emerged. These variations were influenced by cultural, linguistic, and educational

factors. For example, BSL developed in the United Kingdom, ASL in the US and Canada, and Auslan in Australia.

- **Recognition and Standardization:** In the 20th century, sign languages gained recognition as distinct and independent languages. Efforts were made to standardize and document sign languages, Primary to the publication of gesture based language dictionaries and grammars. Linguists recognized that sign languages have their own grammatical rules and structure, separate from spoken languages[27].
- **Deaf Community Activism:** Deaf community activism played a significant role in promoting sign languages and advocating for their recognition. Deaf individuals and organizations fought for the right to use sign languages in education, public services, and other aspects of life. These efforts led to increased awareness and acceptance of sign languages as natural languages with their own linguistic and cultural heritage.
- **Sign Language Legislation:** Many countries have passed legislation to recognize sign languages and ensure the rights of deaf individuals. For instance, countries like New Zealand, Finland, and South Africa have recognized their national sign languages as official languages alongside spoken languages.
- **Globalization and International Sign:** With increased globalization and international communication, the need for a common sign language arose. International Sign (IS) was developed as a pidgin sign language, allowing deaf individuals from different countries to communicate during international events such as conferences and sports tournaments.
- **Manufacture of Sign Language translator:** The profession of gesture based language translator began to take shape in the mid-20th century. In 1943, the first formal sign language interpreter training program was founded in United States at Gallaudet University. The profession expanded as sign language recognition grew, and language translator started functioning in various environment, including educational institutions, courts, and public events.
- **Technological Advancements:** The advent of innovation has significantly Influenced sign speech translation. In the 1980s and 1990s, video relay services

(VRS) emerged, allowing deaf individuals to Convey via gesture based language with hearing individuals over video calls. VRS services often employed sign language interpreters who translated the signed conversation into spoken language and vice versa.

- **Sign Language Recognition Software:** Advancements in AI and computer vision have contributed to the advancement of sign language recognition software. These systems use cameras or sensors to track and interpret the movements of sign language users. While still evolving, these technologies hold promise for real-time sign language translation and the creation of gesture based language avatars or virtual translator.
- **Mobile Applications and Devices:** Today, there are various mobile applications and devices designed to facilitate sign language translation. Some apps aim to interpret verbal language into hand gesture language, while others focus on converting hand gesture language into written or verbal language. These tools provide deaf individuals with greater accessibility and inclusion in everyday communication.

Sign languages continue to evolve and grow, reflecting the cultural and linguistic diversity of deaf communities around the world. They are integral to the identity and communication of deaf individuals, and efforts to promote sign language accessibility and recognition are ongoing.

1.4 Sign Language Translator

A sign to text & speech translator is a type of technology that converts sign language gestures into written or spoken language. This can be done using a variety of methods, such as through computer vision and machine learning algorithms or through sensors that track the movement of the hands and body.[4]

There are a number of sign language to text language translators currently available or in development, which use different approaches to translation. Some rely on pre-programmed gestures and signs[17] while others use machine learning algorithms to interpret a wider range of signs and expressions.

One example of a gestured based language to text language interpreter is the MySign project, which uses sensors to track the hand and finger movements of deaf peoples and translates them into speech & text in the real-time. Other examples include SignSpeak project and the Hand Talk app.

Sign language to text language translators can help to improve communication and accessibility for deaf and Hearing impaired individuals in a variety of environment, from academics and Medical care to employment & social interactions. However, it's important to note that no technology can fully replace the nuance and complexity of human communication and that sign language interpretation by a human translator may still be the most effective method in many situations. [4]

1.4.1 Sign Language Translator Architecture

The architecture indicates the design of physical spaces or environments to be more accessible and inclusive for individuals who have hearing able to hear and rely on signs as their primary mode of communication.

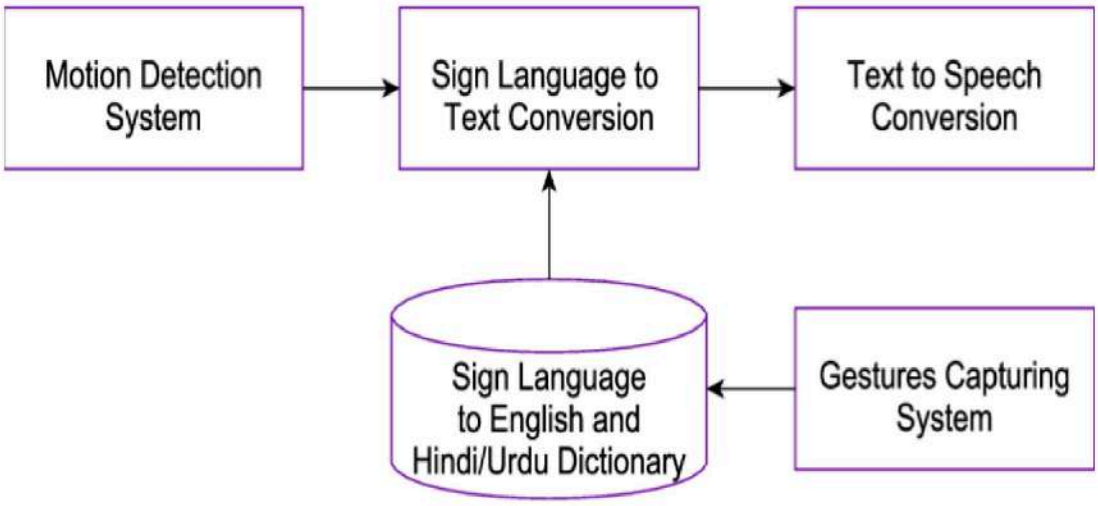

Figure 1.2 Architecture

This can include considerations such as the layout of rooms and buildings, lighting and color schemes, and the placement of furniture and equipment. For example, sign language users often need clear lines of sight to communicate effectively, so designs that

minimize visual obstructions and maximize visibility can be particularly beneficial. In addition to physical design considerations, sign language architecture can also involve the use of technology to facilitate communication. For example, video conferencing software can enable remote interpretation services for sign language users, and smart home devices can be programmed to provide visual alerts for important information such as doorbells and phone calls. Overall, sign language architecture seeks to create more inclusive and accessible environments that enable effective communication and engagement for all individuals, regardless of their hearing ability. [5]

1.4.2 Need Of Sign Language Translator

- **Accessibility**

  Gesture based language serves as the predominant mode of communication for numerous individuals who are deaf or hard of hearing. By employing a sign language to text translator, we can guarantee accessibility of information for this population.

- **Inclusivity**

  With a sign language to text translator, we can create a more inclusive environment for Hearing-impaired individuals, making it easier for them to participate in conversations, meetings, and other social events.

- **Efficiency**

  Sign language to text translator can also help make communication more efficient, especially in situations where there is no interpreter available. It can help save time and reduce miscommunication.

- **Education**

  Sign language to text translator can also be helpful for educators, students, and parents of Hearing-impaired students. It can help bridge the intercommunication gap and ensure that the student receives the same level of education as their peers

1.4.3 Challenges for Sign Language Translator

- **Sign language is not universal**

  Different countries and regions have their own sign languages, which means that a sign language to text translator would need to be tailored to each specific language and dialect.[7]

- **Translation accuracy**

  Sign language is a complex and nuanced language, with many variations in signs and expressions. The accuracy of a sign language to text translator could be affected by the translator's ability to understand and interpret these nuances.[8]

- **Real-time translation**

  Sign language is often used in real-time conversations, and the translator would need to able to keep up with the conversation and translate the signs accurately in real-time.

- **Technical limitations**

  Developing a sign language to text translator that is accurate, reliable, and user-friendly can be a significant technical challenge.[8]

- **Cost**

  Developing and implementing a sign to verbal translator can be expensive, which may limit its accessibility and availability.

Despite these challenges, the manufacturing of a sign language to text translator[23] is important for creating a more inclusive and accessible world for Hearing-impaired individuals. With advancements in technology, these challenges can be overcome, and we can create a more accessible and inclusive world for all.

1.4.4 Working of Sign Language Translator

- **Data collection**

  The first step would be to collect a large dataset of sign language videos and their matching text interpreter. The dataset should include a variety of sign language dialects and expressions to ensure that the translator is accurate and reliable.[9]

- **Data preprocessing**

  The collected data needs to be cleaned and preprocessed. This includes removing any unwanted noise and ensuring that the video and text data are aligned correctly.[9][10]

- **Feature extraction**

  Next, the important features of the sign language videos need to be extracted. This could include hand movements, facial expressions, and body language.[9]

- **Training the model**

  A machine learning model is then trained on the preprocessed data, using algorithms such as deep neural networks, CNN . The model is trained to find the patterns in the gesture language videos or convert them into text.[10][11]

- **Model evaluation**

  Model's performance is evaluated using a separate dataset that was not used for teaching. The evaluation process checks the accuracy and reliability of the translator.[9]

- **Deployment**

  The model has been learned and evaluated, it deployed for use in real-world condition. Translator could be integrated into a mobile app or other communication software.[9]

Overall, developing a language translator to text translator using ML and Python requires a large and diverse dataset, careful data preprocessing, and a well-trained machine learning model. With the right approach, such a translator could help create a more

accessible and inclusive world for deaf and hard of hearing individuals. There are several technologies available that can translate sign language to text. One such technology is computer vision, which uses cameras to capture the movements of the deaf hands and body and then uses machine learning algorithms to translate those movements into text. Another approach is to use sensors worn by the signer to detect their movements, which are then translated into text using machine learning algorithms. There are also devices that use sensors embedded in gloves or other wearable to capture the hand and finger movements of the signer, which are then translated into text.

1.4.5 Application of Language Translator

Education

Sign to text translators can be used in schools and universities to facilitate communication between students who are auditory impairment and teachers who are not proficient in sign language. This can help ensure that students receive the information they need to succeed academically.

Healthcare

Sign language to text translators can help healthcare professionals interact with patients who are auditory impairment. This can improve quality of care provided and ensure that patients fully understand their diagnoses, treatment options, and other health-related information.[18][19]

Business

Sign language to text translators can be used in various business settings, including meetings, conferences, and training sessions, to support employees who are hearing impaired participate fully in these activities.

Customer Service

Sign language to text translators can be used by businesses to provide customer service to customers who are deaf . This can help improve customer satisfaction and loyalty.

Media

Sign language to text translators can be used by media outlets to provide closed subtitles for individuals who are hard of hearing, allowing them to fully access and understand media content.

CHAPTER-2
LITERATURE REVIEW

1. Ais Athania et al, **"RECOGNITION OF SIGN LANGUAGE IN REAL TIME"[9**], in this article, creating communication via gestures application for hard of hearing individuals can be significant, as they'll have the option to discuss effectively with even the individuals who don't comprehend communication via gesture. This article undertaking targets making the fundamental stride in spanning the correspondence hole between ordinary individuals, hard of hearing and unable to speak individuals utilizing sign language. The main focus of this work is to make a vision based system to spot signing gestures. It provides an easier and more intuitive way of communication between a person's and a computer. This work would assist in the achievement of high performance in sign language recognition. It transcribes sign language symbols into plain text and converts the interpreted sign language into speech, allowing real-time communication. This system acts as a link between deaf and mute people and the general public.

2. St. Catharines, **"Research of a Sign Language Translation System Based on Deep Learning"[13]**, this paper discusses the design of hand locating algorithm based on deep learning, the feature extraction based on 3D CNN and the recognition algorithm based on recurrent neural network LSTM, and achieves better recognition results than other methods on common vocabulary data sets.

3. Sakshi Sharma et al**, "Vision-based sign language recognition system: A Comprehensive Review"[**14], the review of existing techniques for recognition of sign language is presented. From the survey presented in this paper, it has observed that an interpreter is required to translate the gesture into text/audio in order to remove the communication gap for the future, work can be done to provide a better and realistic experience for the interface between the signer and nonsigner community.

4. Mohammed Safeel et al**, "Sign Language Recognition Techniques- A Review"[**7], the aim of SLR is to simplify communication between people with impairments and those without, and this manuscript reviews various techniques that have been used for SLR at different stages of recognition. These techniques include image-based methods, segmentation, feature extraction, feature vector quantization, and reduction techniques, as well as classification using training models such as Hidden Markov Models, Deep Learning methods like CNN, k-NN, ANN, SVM, and others. The results and observations from these techniques are compared, and their flexibility allows for their use in major sign detections across various domains.

5. Satwik Ram Kodandaram et al**, "Sign Language Recognition"[**10], this paper aims to enhance automatic recognition of sign language and its translation into text or speech. The paper focuses on identifying static sign language hand gestures such as 26 English alphabets (A-Z) and 10 digits (0-9) using Deep Neural Networks (DNN). A convolution neural networks classifier was created and trained using different configurations and architectures such as LeNet-5, MobileNetV2, and a custom architecture to achieve maximum accuracy. Then they developed a web application using Django Rest Frameworks to test results from a live camera. 10

6. G Arun Prasath et al, **"Prediction of sign language recognition based on multi layered CNN"[**15], in this paper the process transforms sign language into the voice for assisting the people to hear the sign language. The ROBITA Indian Sign Language Gesture Database is the source of input data, and certain necessary pre-processing measures are taken to prevent unwanted artifacts. To evaluate the scalability and accuracy of end-to-end SLR, the suggested approach incorporates an encoder Multi-Layer Convolutional Neural Network (ML-CNN). The encoder examines both linear and nonlinear features, both higher and lower levels, in order to improve recognition quality. The simulation is conducted in MATLAB, where the ML-CNN model's performance outperforms existing methods and establishes a trade-off. The limitation with this model is the computational complexity with the slightly higher. As future research directions, the analysis can be carried out with the real-

time video sequences over the complex environment.

7. Harini R et al, S**, "Sign Language Translation"[**8], the aim of the initiative is to build a visual reorganization system that can instantly recognize and translate users' hand signals into text. The system comprises four key components: image capturing, pre-processing, classification, and prediction. Through image processing, the system can segment the input signals, which are then captured and analyzed using the OpenCV Python library. The captured signals are resized, turned into grayscale images, and filtered to reduce noise and enhance prediction accuracy. Finally, the system utilizes a convolution neural network to classify and predict the input signals. Although the facial expressions express a lot during communication, the system does not focus on facial expressions. The accuracy of the model was less with poor lighting. As future enhancements, more dynamic video signs can be trained involving facial features and expressions.

8. Varsha M et al**, "Sign Language Gesture Recognition Using Deep Convolutional Neural Network"[**5], the aim of this work is to recognize ISL gestures and convert it into text. Currently, an image recognition model was implemented using deep CNN (Inception V3 model) which accepts input image and it is passed through a series of layers and the output is generated. It has been found that the system produced better results with a smaller number of epochs. The future work is to translate the recognized gesture to text. The work can also be extended to recognize continuous sign language gestures from real-time videos.

9. Hsien-I Lin , Ming-Hsiang Hsu, and Wei-Kai, **"Human Hand Gesture Recognition Using a Convolution Neural Network"**[28] **,** they construct a skin model to extract the hand out of an image and then apply binary threshold to the whole image. After obtaining the threshold image they calibrate it about the principal axis in order to center the image about it. They input this image to a convolutional neural network model in order to train and predict the outputs. They have trained their model over 7 hand gestures and using their model they produce an accuracy of around 95% for those 7 gestures.

2.1 PROBLEM STATEMENT

- Sign language recognition is a study area with wide applications and build potential, but the data collection is finite in scope.
- The challenges of uniformity, regional differences, and data acquisition must be overcome to design an efficient system. Despite decades of research, creating a sign language translator remains a difficult task due to variations in sign appearance depending on the signer and viewpoint.
- Nonetheless, sign language recognition offers an opportunity to remove communication barriers and develop better techniques using existing literature.

CHAPTER-3

METHODOLOGY

3.1 Working of Sign Language Translator

Data collection: The first step would be to collect a large dataset of sign language videos and their corresponding text translations. The dataset should include a variety of sign language

Data preprocessing: The collected data needs to be cleaned and preprocessed. This includes removing any unwanted noise and ensuring that the video and text data are aligned correctly.[9][10]

Feature extraction: Next, the important features of the sign language videos need to be extracted. This could include hand movements, facial expressions, and body language.[9]

Training the model: A machine learning model is then trained on the preprocessed data, using algorithms such as deep neural networks, CNN. The model is trained to recognize patterns in the sign language videos and translate them into text. A CNN typically has three layers: a convolutional layer, pooling layer, and fully connected layer.[10][11]

3.1.1 Convolutional layer

The main aim is to extract feature i.e. edges, colours & corners from the input. As we read deeply the network, it starts searching more complex characteristics i.e. shapes, face parts as well.[10][11]

Pooling Layer:

This is used to decrease the computational power which we need to process the data. It is done by minimizing the dimensions of the matrix. In this, we tries to extract the dominant features from a restricted amount of neighborhood. [10][11]

Fully connected layer:

Till now we haven't done anything about classifying different images, what we have done is highlighted some features in an image and reduces the dimensions of the image drastically.[10][11]

Model evaluation:

The model's performance is evaluated using a separate dataset that was not used for training. The evaluation process checks the accuracy and reliability of the translator.[9]

Deployment:

Once the model has been trained and evaluated, it can be deployed for use in real-world situations[22]. The translator could be integrated into a mobile app or other communication software.[9]

Overall, developing a sign language to text translator using machine learning[21] and Python requires a large and diverse dataset, careful data preprocessing, and a well-trained machine learning model. With the right approach, such a translator could help create a more accessible and inclusive world for deaf and hard of hearing individuals[21]. There are several technologies available that can translate sign language to text. One such technology is computer vision, which uses cameras to capture the movements of the signer's hands and body and then uses machine learning algorithms to translate those movements into text. Another approach is to use sensors worn by the signer to detect their movements, which are then translated into text using machine learning algorithms. There are also devices that use sensors embedded in gloves or other wearables to capture the hand and finger movements of the signer, which are then translated into text.

3.2 Technologies Used

3.2.1 Visual Studio code

VS Code is highly customizable source code editor which was developed by Microsoft. It offers a UI and a wide range of features, which includes syntax highlighting, code

completion & integration with version control systems like Git. VS Code supports multiple programming languages and offers a rich extension ecosystem that allows developers to increase the editor's functionality according to their specific needs. It is available for Windows and Linux, making it usable to a broad range of developers.

Environment of VS code

It is the first screen environment of VS code, where we can run our code & set the parameters which are being used.

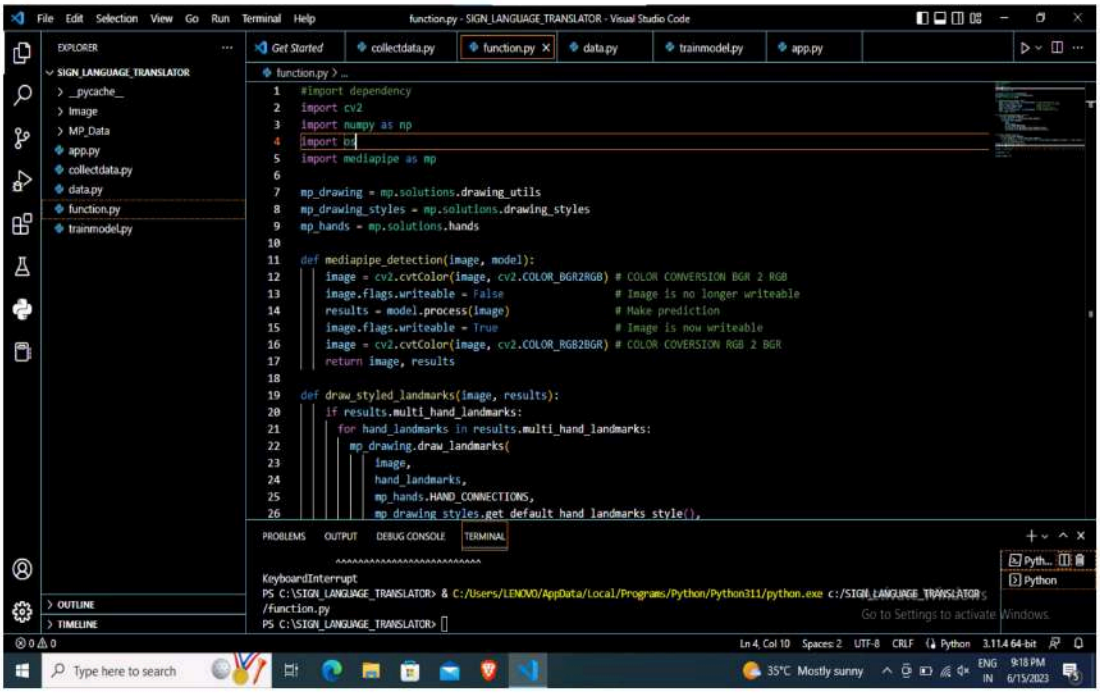

Figure 3.1 VS Code Environment

3.2.2 OpenCV

OpenCV is an open-source computer vision and ml software library. It offers a comprehensive set of tools and techniques for image, video processing and more. OpenCV is widely used in various robotics, facial recognition, and medical image analysis.

Features:

- Image & Video Processing
- Feature Detection & Description
- Object Detection & Tracking
- ML Integration
- Camera Calibration & 3D Vision
- Cross-Platform & Language Support

3.2.3 MediaPipe

MediaPipe is an open-source framework which was found by Google which enables the real-time multimedia processing pipelines's construction. It offers a set of pre-built components and tools for building applications that process and analyze audio, video, and sensor data. MediaPipe simplifies the development of complex multimedia applications by abstracting away low-level details and offering a high-level interface for creating real-time processing pipelines.

Features:

- Graph-based Pipeline
- Pre-built Components
- Cross-platform Support
- Integration with Machine Learning'

3.2.4 TensorFlow

TensorFlow is an open-source ml framework which was developed by Google. It offers a comprehensive ecosystem of technologies, tools, libraries & resources for building & deploying ml models. TensorFlow is designed to be scalable, and efficient & making it suitable for a wide range of applications, from research to production deployment.

3.2.5 Keras

Keras is a high-level deep learning library which is widely used for building and training neural networks[22]. It provides a user-friendly and intuitive interface to define, compile, and train various types of deep learning models. Keras has gained popularity in the deep learning community because to its simplicity, flexibility, and compatibility with popular deep learning frameworks. It is often used for a wide range of tasks, including image classification, object detection, natural language processing, and more.

Features of Keras:

- User-friendly API
- Neural network building blocks
- Model compilation
- Training and evaluation
- GPU support
- Model saving and loading
- Transfer learning

CHAPTER-4

OBJECTIVE

- The objective of this project is to contribute to the field of automatic gesture language recognition and translation to text or speech.
- This work focused on recognizing the hand gestures which includes A-Z and 0-9 using DNN.
- We created a convolution neural networks classifier which can classified the gestures into English alphabets and digits. We have trained the neural network under different configurations and architectures

CODE:

1. DATA COLECTION

```
import os

import cv2

cap=cv2.VideoCapture(0)

directory='Image/'

while True:

_,frame=cap.read()

count = {

'a': len(os.listdir(directory+"/A")),

'b': len(os.listdir(directory+"/B")),

'c': len(os.listdir(directory+"/C")),

'd': len(os.listdir(directory+"/D")),

'e': len(os.listdir(directory+"/E")),
```

```
'f': len(os.listdir(directory+"/F")),
'g': len(os.listdir(directory+"/G")),
'h': len(os.listdir(directory+"/H")),
'i': len(os.listdir(directory+"/I")),
'j': len(os.listdir(directory+"/J")),
'k': len(os.listdir(directory+"/K")),
'l': len(os.listdir(directory+"/L")),
'm': len(os.listdir(directory+"/M")),
'n': len(os.listdir(directory+"/N")),
'o': len(os.listdir(directory+"/O")),
'p': len(os.listdir(directory+"/P")),
'q': len(os.listdir(directory+"/Q")),
'r': len(os.listdir(directory+"/R")),
's': len(os.listdir(directory+"/S")),
't': len(os.listdir(directory+"/T")),
'u': len(os.listdir(directory+"/U")),
'v': len(os.listdir(directory+"/V")),
'w': len(os.listdir(directory+"/W")),
'x': len(os.listdir(directory+"/X")),
'y': len(os.listdir(directory+"/Y")),
'z': len(os.listdir(directory+"/Z")),
'0': len(os.listdir(directory+"/0")),
'1': len(os.listdir(directory+"/1")),
```

```
'2': len(os.listdir(directory+"/2")),
'3': len(os.listdir(directory+"/3")),
'4': len(os.listdir(directory+"/4")),
'5': len(os.listdir(directory+"/5")),
'6': len(os.listdir(directory+"/6")),
'7': len(os.listdir(directory+"/7")),
'8': len(os.listdir(directory+"/8")),
'9': len(os.listdir(directory+"/9")),
}
row = frame.shape[1]
col = frame.shape[0]
cv2.rectangle(frame,(0,40),(300,400),(255,255,255),2)
cv2.imshow("data",frame)
cv2.imshow("ROI",frame[40:400,0:300])
frame=frame[40:400,0:300]
interrupt = cv2.waitKey(10)
if interrupt & 0xFF == ord('a'):
cv2.imwrite(directory+'A/'+str(count['a'])+'.png',frame)
if interrupt & 0xFF == ord('b'):
cv2.imwrite(directory+'B/'+str(count['b'])+'.png',frame)
if interrupt & 0xFF == ord('c'):
cv2.imwrite(directory+'C/'+str(count['c'])+'.png',frame)
if interrupt & 0xFF == ord('d'):
```

```
cv2.imwrite(directory+'D/'+str(count['d'])+'.png',frame)
if interrupt & 0xFF == ord('e'):
cv2.imwrite(directory+'E/'+str(count['e'])+'.png',frame)
if interrupt & 0xFF == ord('f'):
cv2.imwrite(directory+'F/'+str(count['f'])+'.png',frame)
if interrupt & 0xFF == ord('g'):
cv2.imwrite(directory+'G/'+str(count['g'])+'.png',frame)
if interrupt & 0xFF == ord('h'):
cv2.imwrite(directory+'H/'+str(count['h'])+'.png',frame)
if interrupt & 0xFF == ord('i'):
cv2.imwrite(directory+'I/'+str(count['i'])+'.png',frame)
if interrupt & 0xFF == ord('j'):
cv2.imwrite(directory+'J/'+str(count['j'])+'.png',frame)
if interrupt & 0xFF == ord('k'):
cv2.imwrite(directory+'K/'+str(count['k'])+'.png',frame)
if interrupt & 0xFF == ord('l'):
cv2.imwrite(directory+'L/'+str(count['l'])+'.png',frame)
if interrupt & 0xFF == ord('m'):
cv2.imwrite(directory+'M/'+str(count['m'])+'.png',frame)
if interrupt & 0xFF == ord('n'):
cv2.imwrite(directory+'N/'+str(count['n'])+'.png',frame)
if interrupt & 0xFF == ord('o'):
cv2.imwrite(directory+'O/'+str(count['o'])+'.png',frame)
```

```
if interrupt & 0xFF == ord('p'):
cv2.imwrite(directory+'P/'+str(count['p'])+'.png',frame)
if interrupt & 0xFF == ord('q'):
cv2.imwrite(directory+'Q/'+str(count['q'])+'.png',frame)
if interrupt & 0xFF == ord('r'):
cv2.imwrite(directory+'R/'+str(count['r'])+'.png',frame)
if interrupt & 0xFF == ord('s'):
cv2.imwrite(directory+'S/'+str(count['s'])+'.png',frame)
if interrupt & 0xFF == ord('t'):
cv2.imwrite(directory+'T/'+str(count['t'])+'.png',frame)
if interrupt & 0xFF == ord('u'):
cv2.imwrite(directory+'U/'+str(count['u'])+'.png',frame)
if interrupt & 0xFF == ord('v'):
cv2.imwrite(directory+'V/'+str(count['v'])+'.png',frame)
if interrupt & 0xFF == ord('w'):
cv2.imwrite(directory+'W/'+str(count['w'])+'.png',frame)
if interrupt & 0xFF == ord('x'):
cv2.imwrite(directory+'X/'+str(count['x'])+'.png',frame)
if interrupt & 0xFF == ord('y'):
cv2.imwrite(directory+'Y/'+str(count['y'])+'.png',frame)
if interrupt & 0xFF == ord('z'):
cv2.imwrite(directory+'Z/'+str(count['z'])+'.png',frame)
if interrupt & 0xFF == ord('1'):
```

```
cv2.imwrite(directory+'1/'+str(count['1'])+'.png',frame)
if interrupt & 0xFF == ord('2'):
cv2.imwrite(directory+'2/'+str(count['2'])+'.png',frame)
if interrupt & 0xFF == ord('3'):
cv2.imwrite(directory+'3/'+str(count['3'])+'.png',frame)
if interrupt & 0xFF == ord('4'):
cv2.imwrite(directory+'4/'+str(count['4'])+'.png',frame)
if interrupt & 0xFF == ord('5'):
cv2.imwrite(directory+'5/'+str(count['5'])+'.png',frame)
if interrupt & 0xFF == ord('6'):
cv2.imwrite(directory+'6/'+str(count['6'])+'.png',frame)
if interrupt & 0xFF == ord('7'):
cv2.imwrite(directory+'7/'+str(count['7'])+'.png',frame)
if interrupt & 0xFF == ord('8'):
cv2.imwrite(directory+'8/'+str(count['8'])+'.png',frame)
if interrupt & 0xFF == ord('9'):
cv2.imwrite(directory+'9/'+str(count['9'])+'.png',frame)
if interrupt & 0xFF == ord('0'):
cv2.imwrite(directory+'0/'+str(count['0'])+'.png',frame)
cap.release()
cv2.destroyAllWindows()
```

OUTPUT:

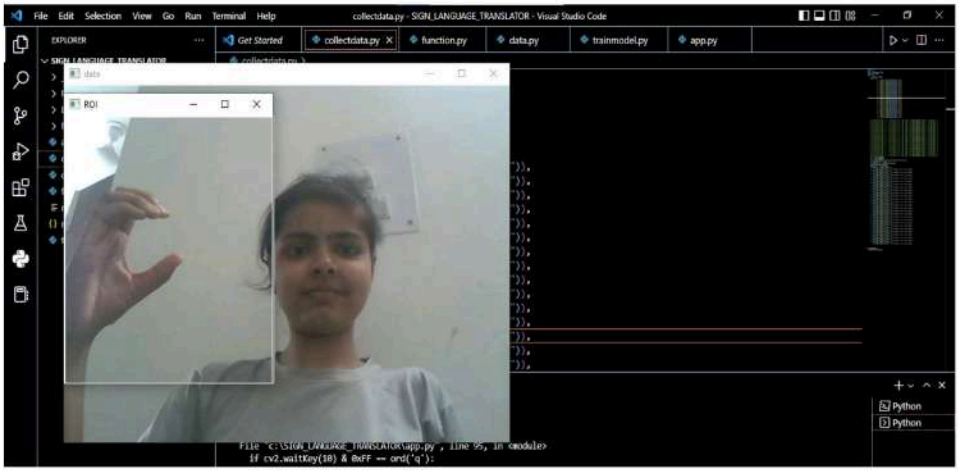

Figure 4.1 Output of data.py

The above code captures video from the default camera and allows us to save frames as images based on the keys we press. The frames are saved in different directories based on the key pressed. Here's a breakdown of the code:

1. The code starts by importing the necessary libraries: os and cv2 (OpenCV).
2. It initializes the video capture object using cv2.VideoCapture(0), which captures video from the default camera (index 0).
3. It sets the directory where the images will be saved using the directory variable by making sure the directory exists before running the code.
4. Inside the while loop, it reads frames from the video capture object using cap.read(). The underscore _ is used to discard the first value returned by cap.read(), which indicates whether the frame was successfully read.
5. It defines a dictionary count that keeps track of the number of images saved in each directory.
6. The frame is displayed using cv2.imshow().
7. A rectangle is drawn on the frame using cv2.rectangle().

8. A region of interest (ROI) is selected by slicing the frame using frame[40:400, 0:300].
9. The program waits for a key press using cv2.waitKey(10).
10. If a key is pressed and matches one of the defined keys ('a' to 'z'), the corresponding image is saved. For example, if the 'a' key is pressed, the ROI frame is saved in the 'A' directory with a filename based on the count.
11. The cap.release() method is called to release the video capture object.
12. Finally, the windows are closed using cv2.destroyAllWindows().

Note: The code will run indefinitely until manually interrupted. We have to make sure to terminate the program when we are done capturing images. We need to create the necessary directories ('A' to 'Z') inside the 'Image' directory before running the code.

2. Functioning

```
#import dependency

import cv2

import numpy as np

import os

import mediapipe as mp

mp_drawing = mp.solutions.drawing_utils

mp_drawing_styles = mp.solutions.drawing_styles

mp_hands = mp.solutions.hands

def mediapipe_detection(image, model):

    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB) # COLOR CONVERSION BGR 2 RGB

    image.flags.writeable = False                  # Image is no longer writeable
```

```
    results = model.process(image)                # Make prediction
    image.flags.writeable = True                  # Image is now writeable
    image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR) # COLOR COVERSION RGB 2 BGR
    return image, results
def draw_styled_landmarks(image, results):
    if results.multi_hand_landmarks:
      for hand_landmarks in results.multi_hand_landmarks:
        mp_drawing.draw_landmarks(
            image,
            hand_landmarks,
            mp_hands.HAND_CONNECTIONS,
            mp_drawing_styles.get_default_hand_landmarks_style(),
            mp_drawing_styles.get_default_hand_connections_style())
def extract_keypoints(results):
    if results.multi_hand_landmarks:
      for hand_landmarks in results.multi_hand_landmarks:
        rh = np.array([[res.x, res.y, res.z] for res in hand_landmarks.landmark]).flatten() if hand_landmarks else np.zeros(21*3)
        return(np.concatenate([rh]))
# Path for exported data, numpy arrays
DATA_PATH = os.path.join('MP_Data')
```

actions = np.array(['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','0','1','2','3','4','5','6','7','8','9'])

no_sequences = 30

sequence_length = 30

OUTPUT:

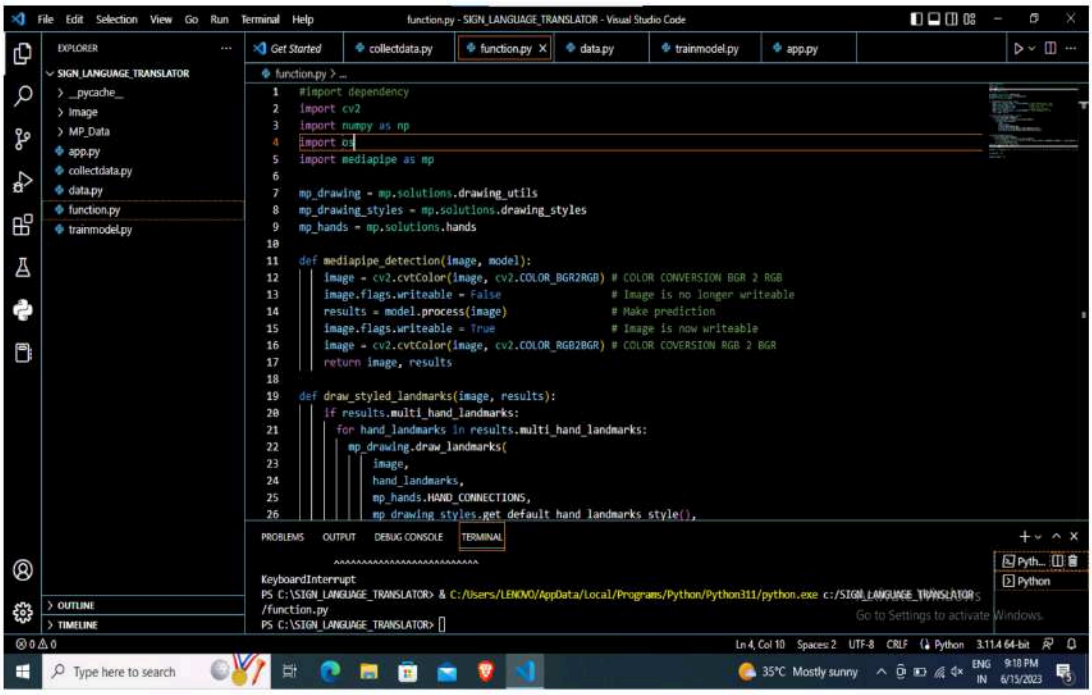

Figure 4.2 function.py file

The above code is using the Mediapipe library to perform hand detection and landmark extraction from an image. It also includes some variables related to data storage and defines an array of actions.

Here's a breakdown of the code:

1. The necessary libraries are imported: cv2, numpy, os, and mediapipe.
2. The mediapipe_detection function takes an image and a model as input. It converts the image from BGR to RGB format, processes it with the model, and

then converts it back to BGR format before returning the image and the detection results.

3. The draw_styled_landmarks function takes an image and the detection results as input. It draws landmarks and connections on the image using the mp_drawing.draw_landmarks function.
4. The extract_keypoints function takes the detection results as input. It extracts the 3D coordinates (x, y, z) of the hand landmarks and returns them as a flattened array.
5. The DATA_PATH variable is defined to specify the path where the exported data (numpy arrays) will be stored.
6. An array actions is created, containing the labels for the different hand gestures or actions.
7. The no_sequences variable specifies the number of sequences (samples) that will be recorded for each action.
8. The sequence_length variable specifies the length of each sequence (number of frames).

3. data.py

```
from function import *
from time import sleep
for action in actions:
    for sequence in range(no_sequences):
        try:
            os.makedirs(os.path.join(DATA_PATH, action, str(sequence)))
        except:
            pass
```

```
# cap = cv2.VideoCapture(0)
# Set mediapipe model
with mp_hands.Hands(
    model_complexity=0,
    min_detection_confidence=0.5,
    min_tracking_confidence=0.5) as hands:
    # NEW LOOP
    # Loop through actions
    for action in actions:
        # Loop through sequences aka videos
        for sequence in range(no_sequences):
            # Loop through video length aka sequence length
            for frame_num in range(sequence_length):
                # Read feed
                frame=cv2.imread('Image/{}/{}.png'.format(action,sequence))
                # Make detections
                image, results = mediapipe_detection(frame, hands)
               #print(results)
                # Draw landmarks
                draw_styled_landmarks(image, results)
                # NEW Apply wait logic
                if frame_num == 0:
                    cv2.putText(image, 'STARTING COLLECTION', (120,200),
```

```
                    cv2.FONT_HERSHEY_SIMPLEX, 1, (0,255, 0), 4,
cv2.LINE_AA)
                cv2.putText(image, 'Collecting frames for {} Video Number
{}'.format(action, sequence), (15,12),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 1,
cv2.LINE_AA)
                # Show to screen
                cv2.imshow('OpenCV Feed', image)
                cv2.waitKey(200)
            else:
                cv2.putText(image, 'Collecting frames for {} Video Number
{}'.format(action, sequence), (15,12),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 1,
cv2.LINE_AA)
                # Show to screen
                cv2.imshow('OpenCV Feed', image)
            # NEW Export keypoints
            keypoints = extract_keypoints(results)
            npy_path = os.path.join(DATA_PATH, action, str(sequence),
str(frame_num))
            np.save(npy_path, keypoints)
            # Break gracefully
            if cv2.waitKey(10) & 0xFF == ord('q'):
                break
```

```
# cap.release()

cv2.destroyAllWindows()
```

OUTPUT:

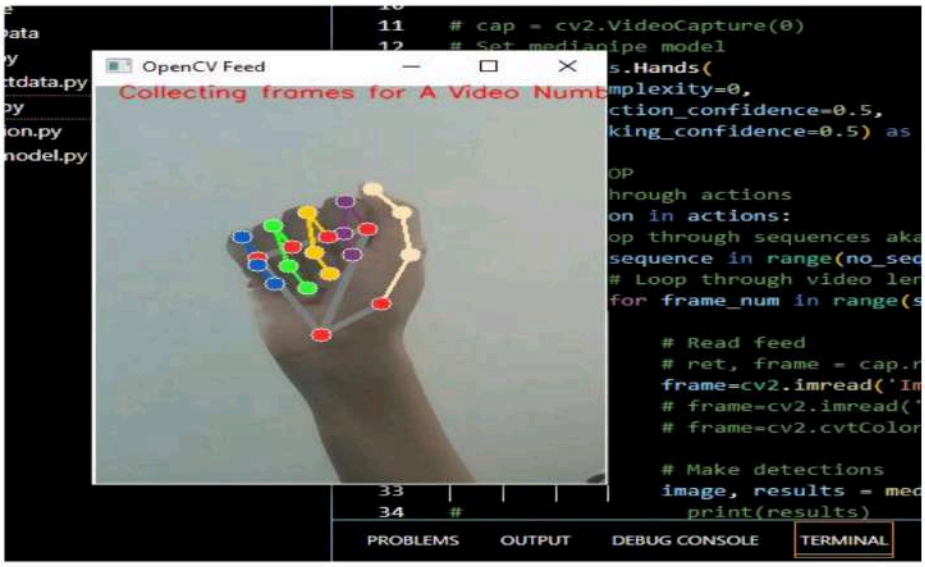

Figure 4.3 Output of data.py

The above code seems to continue from where we left off. It imports the necessary functions and modules, defines some variables, and then goes into the main loop to capture and process hand gesture data using Mediapipe.

Here's a breakdown of the code:

1. It imports the **sleep** function from the **time** module.
2. The outer loop iterates through each action in the **actions** array.
3. The next inner loop iterates through each sequence number (video) for the current action.
4. The next inner loop iterates through each frame within the sequence.
5. It reads an image/frame from a file using **cv2.imread('Image/{}/{}.png'.format(action,sequence))**.

6. The **mediapipe_detection** function is called to process the frame and obtain the detection results.
7. The **draw_styled_landmarks** function is called to draw landmarks on the image.
8. Conditional statements and text annotations are added to the image to indicate the start of collection and the action and sequence being recorded.
9. The image is displayed using **cv2.imshow**().
10. Keypoints are extracted from the detection results using the **extract_keypoints** function.
11. The keypoints are saved as a numpy array using **np.save**().
12. The loop breaks if the 'q' key is pressed.
13. After the main loop, the video capture object is released and the windows are closed.

4. Training Model

```
from function import *
from sklearn.model_selection import train_test_split
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import LSTM, Dense
from keras.callbacks import TensorBoard
label_map = {label:num for num, label in enumerate(actions)}
# print(label_map)
sequences, labels = [], []
for action in actions:
```

```
    for sequence in range(no_sequences):
        window = []
        for frame_num in range(sequence_length):
            res = np.load(os.path.join(DATA_PATH, action, str(sequence), "{}.npy".format(frame_num)))
            window.append(res)
        sequences.append(window)
        labels.append(label_map[action])
X = np.array(sequences)
y = to_categorical(labels).astype(int)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.05)
log_dir = os.path.join('Logs')
tb_callback = TensorBoard(log_dir=log_dir)
model = Sequential()
model.add(LSTM(64, return_sequences=True, activation='relu', input_shape=(30,63)))
model.add(LSTM(128, return_sequences=True, activation='relu'))
model.add(LSTM(64, return_sequences=False, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(actions.shape[0], activation='softmax'))
res = [.7, 0.2, 0.1]
```

```
model.compile(optimizer='Adam', loss='categorical_crossentropy',
metrics=['categorical_accuracy'])

model.fit(X_train, y_train, epochs=200, callbacks=[tb_callback])

model.summary()

model_json = model.to_json()

with open("model.json", "w") as json_file:

   json_file.write(model_json)

model.save('model.h5')
```

OUTPUT:

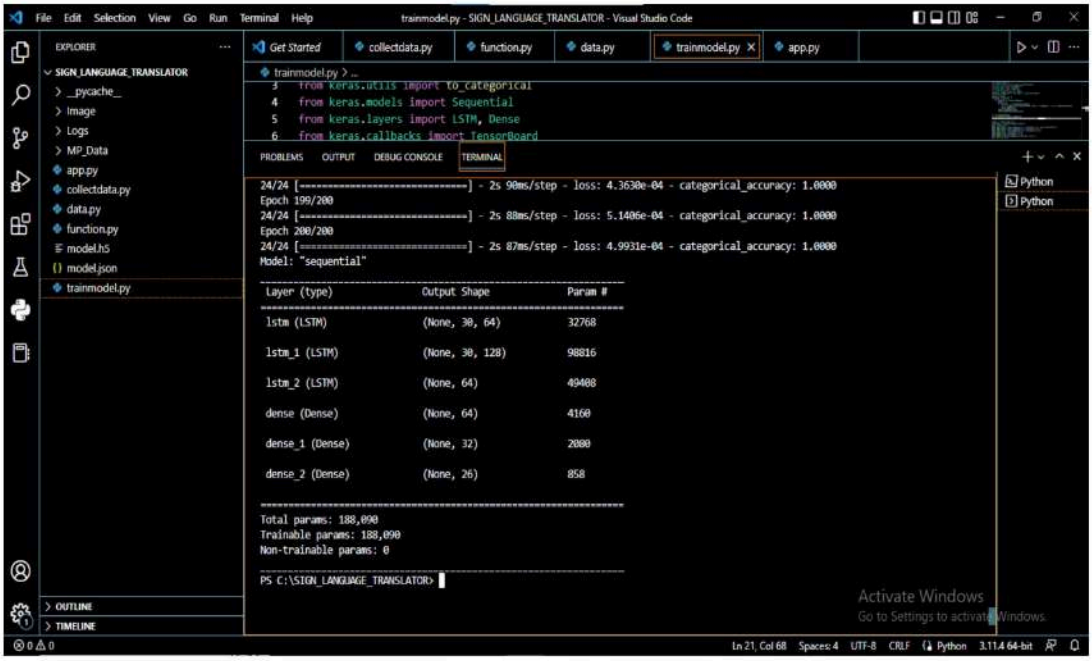

Figure 4.4 Output of trainmodel.py

The above code focuses on training a deep learning model using the captured hand gesture data. Here's a breakdown of the code:

1. It imports additional necessary functions and modules, including **train_test_split** from **sklearn.model_selection**, **to_categorical** from **keras.utils**, **Sequential** and **Dense** from **keras.layers**, and **TensorBoard** from **keras.callbacks**.
2. A label map is created to map each action label to a numerical value.
3. Two lists, **sequences** and **labels**, are initialized to store the input sequences and their corresponding labels.
4. Nested loops iterate through each action and sequence to load the saved keypoints from numpy files and construct the input sequences.
5. The constructed sequences and their labels are appended to the **sequences** and **labels** lists.
6. The sequences and labels are converted to numpy arrays, and the labels are one-hot encoded using **to_categorical**.
7. The data is split into training and testing sets using **train_test_split**.
8. The log directory for TensorBoard is specified, and a TensorBoard callback is created.
9. The model architecture is defined using the **Sequential** API of Keras. It consists of LSTM layers followed by dense layers. The input shape is specified based on the sequence length and the number of features in each frame.
10. The model is compiled with the Adam optimizer, categorical cross-entropy loss, and metrics to track categorical accuracy.
11. The model is trained using the training data for a specified number of epochs, and the TensorBoard callback is provided.
12. The model summary is printed.

13. The model is saved in two formats: as a JSON file (**model.json**) and as a complete model file (**model.h5**).

5. Final Output

```
from function import *
from keras.utils import to_categorical
from keras.models import model_from_json
from keras.layers import LSTM, Dense
from keras.callbacks import TensorBoard
json_file = open("model.json", "r")
model_json = json_file.read()
json_file.close()
model = model_from_json(model_json)
model.load_weights("model.h5")
colors = []
for i in range(0,20):
    colors.append((245,117,16))
print(len(colors))
def prob_viz(res, actions, input_frame, colors,threshold):
    output_frame = input_frame.copy()
    for num, prob in enumerate(res):
        cv2.rectangle(output_frame, (0,60+num*40), (int(prob*100), 90+num*40),
colors[num], -1)
```

```
        cv2.putText(output_frame, actions[num], (0, 85+num*40),
cv2.FONT_HERSHEY_SIMPLEX, 1, (255,255,255), 2, cv2.LINE_AA
    return output_frame
# 1. New detection variables
sequence = []
sentence = []
accuracy=[]
predictions = []
threshold = 0.8
cap = cv2.VideoCapture(0)
# Set mediapipe model
with mp_hands.Hands(
    model_complexity=0,
    min_detection_confidence=0.5,
    min_tracking_confidence=0.5) as hands:
    while cap.isOpened():
        # Read feed
        ret, frame = cap.read()
        # Make detections
        cropframe=frame[40:400,0:300]
        # print(frame.shape)
        frame=cv2.rectangle(frame,(0,40),(300,400),255,2)
```

```
        #frame=cv2.putText(frame,"Active
Region",(75,25),cv2.FONT_HERSHEY_COMPLEX_SMALL,2,255,2)
        image, results = mediapipe_detection(cropframe, hands)
        # print(results)
        # Draw landmarks
        # draw_styled_landmarks(image, results)
        # 2. Prediction logic
        keypoints = extract_keypoints(results)
        sequence.append(keypoints)
        sequence = sequence[-30:]
        try:
            if len(sequence) == 30:
                res = model.predict(np.expand_dims(sequence, axis=0))[0]
                print(actions[np.argmax(res)])
                predictions.append(np.argmax(res))
            #3. Viz logic
                if np.unique(predictions[-10:])[0]==np.argmax(res):
                    if res[np.argmax(res)] > threshold:
                        if len(sentence) > 0:
                            if actions[np.argmax(res)] != sentence[-1]:
                                sentence.append(actions[np.argmax(res)])
                                accuracy.append(str(res[np.argmax(res)]*100))
                        else:
```

```
                    sentence.append(actions[np.argmax(res)])
                    accuracy.append(str(res[np.argmax(res)]*100))
            if len(sentence) > 1:
                sentence = sentence[-1:]
                accuracy=accuracy[-1:]
            # Viz probabilities
            # frame = prob_viz(res, actions, frame, colors,threshold)
        except Exception as e:
            # print(e)
            pass
        cv2.rectangle(frame, (0,0), (300, 40), (245, 117, 16), -1)
        cv2.putText(frame,"Output: -"+' '.join(sentence)+".join(accuracy), (3,30),
                    cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2,
cv2.LINE_AA)
        # Show to screen
        cv2.imshow('OpenCV Feed', frame)
        # Break gracefully
        if cv2.waitKey(10) & 0xFF == ord('q'):
            break
    cap.release()
    cv2.destroyAllWindows()
```

OUTPUT:

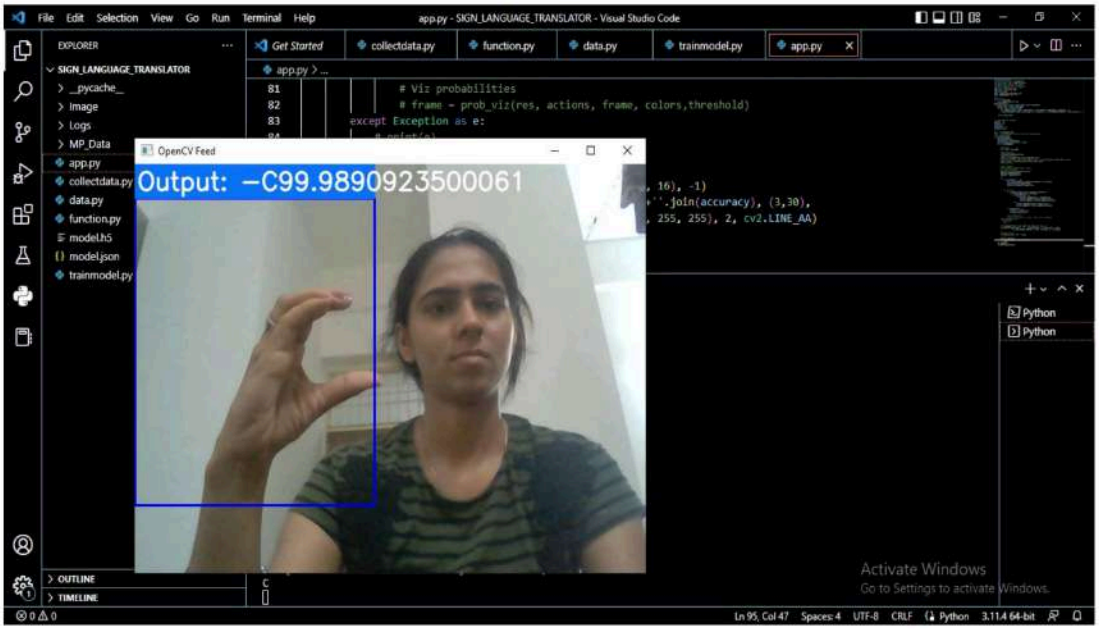

Figure 4.5 Output of app.py

The above code focuses on real-time hand gesture recognition using the trained model. Here's a breakdown of the code:

1. The code imports additional necessary functions and modules, including to_categorical from keras.utils, model_from_json from keras.models, and LSTM, Dense, and TensorBoard from keras.layers.

2. The code loads the trained model from the saved JSON file (model.json) and its weights (model.h5).

3. A list of colors is initialized to visualize the probabilities of different actions.

4. The prob_viz function is defined to visualize the probabilities as rectangles on the output frame.

5. New variables sequence, sentence, accuracy, and predictions are initialized to store the input sequence, recognized sentence, prediction accuracy, and predicted actions.

6. The code initializes the video capture using cv2.VideoCapture.

7. The video frames are read in a loop using cap.read().

8. The frame is cropped to focus on the hand region, and hand detection is performed using the Mediapipe model.

9. The keypoints are extracted from the hand detection results.

10. The keypoints are added to the sequence list, and only the last 30 keypoints are kept in the sequence.

11. The model predicts the action label based on the input sequence, and the predicted label is appended to the predictions list.

12. If the predicted label remains consistent for the last 10 predictions and surpasses a specified threshold, it is considered as the recognized action. The recognized action is appended to the sentence list, along with the corresponding prediction accuracy in the accuracy list.

13. Visualization logic is commented out in this code. It was used to visualize the probabilities of different actions on the frame.

14. The recognized sentence and accuracy are displayed on the frame.

15. The frame is shown on the screen using cv2.imshow, and the loop continues until the user presses 'q' to break out of the loop.

16. The video capture is released, and all windows are closed using cap.release() and cv2.destroyAllWindows().

CHAPTER-5

CONCLUSION

In this study, the development of a gesture language to text translator has brought about a remarkable breakthrough in facilitating communication between sign language users and individuals who rely on text-based communication. It empowers them to engage in conversations and interactions with a wider range of people, eliminating communication barriers and promoting inclusivity. It promotes efficient and effective conversations, fostering better understanding and connection. It has the potential to make bridge the gap between gesture language and text-based communication, promoting inclusivity, independence, and equal opportunities. Continued efforts in its development and refinement will further amplify its impact and benefit the global community.

5.1 FUTURE SCOPE:

- We focus only on finger signs, but sign language also involves facial and body expressions. While the system should be used as a module for more advanced work, further efforts are needed for a complete solution.

- The future scope for sign language translator technology is very promising. With advances in artificial intelligence and natural language processing, there is great potential for developing highly accurate and intuitive sign language translation systems.

- One major area where sign language translator technology could be highly beneficial is in education. Many deaf and hard-of-hearing students struggle to access educational materials that are designed for hearing individuals. A sign language translator could help to bridge this gap by enabling these students to better understand lectures, videos, and other educational content.

- Another potential application of sign language translator technology is in the workplace. Many deaf and hard-of-hearing individuals face communication barriers in the workplace, which can limit their career opportunities.

REFERENCE

[1] https://www.computerhope.com/jargon/t/translator.htm

[2] https://en.wikipedia.org/wiki/Statistical_machine_translation

[3] https://www.memorialhearing.com/

[4] https://edubirdie.com/examples/sign-language-interpretation-using-deep-learning/

[5] Varsha M, Chitra S Nair, "Indian Sign Language Gesture Recognition Using Deep Convolutional Neural Network," *8th International Conference on Smart Computing and Communications (ICSCC)*, 978-1-7281-9687-9/21 IEEE, doi: 10.1109/ICSCC51209.2021.9528246, 2021

[6] https://www.istockphoto.com/photos/sign-language-alphabet

[7] Mohammed Safeel, Tejas Sukumar, Shashank K S, Arman M D, Shashidhar R, Puneeth SB, "Sign Language Recognition Techniques- A Review," *IEEE International Conference for Innovation in Technology (INOCON)* 978-1-7281-9744-9/20 IEEE, 2020

[8] Harini R, Janani R, Keerthana S, Madhubala S, Venkatasubramanian S, "Sign Language Translation," *6th International Conference on Advanced Computing & Communications Systems (ICACCS)* 978-1-7281-5197-7/20 IEEE, 2020

[9] Ais Athania, Krishna Sanjay Gupta, Karima Khan and A. E. Patil, "Recognition of Sign Language in Real Time," *International Journal for Research in Engineering Application & Management* (ISSN: 2454-9150) Vol-07, APR 2021, doi: 10.35291/2454-9150.2021.0130 https://www.researchgate.net/publication/369142397, APR 2021

[10] Satwik Ram Kodandaram, N Pavan Kumar, Sunil G L, "Sign Language Recognition," doi: 10.13140/RG.2.2.29061.47845, 2021 https://www.researchgate.net/publication/353141966

[11] https://medium.com/nybles/a-brief-guide-to-convolutional-neural-network-cnn642f47e88ed4

[12] https://miro.medium.com/v2/resize:fit:1400/format:webp/1*F2Ik_XFzmu5jZFbyiAKQQ.jpeg

[13] St. Catharines, "Research of a Sign Language Translation System Based on Deep Learning," *International Conference on Artificial Intelligence and Advanced Manufacturing (AIAM)*, doi: 10.1109/AIAM48774.2019.00083, 2019

[14] Sakshi Sharma, Sukhwinder Singh, "Vision-based sign language recognition system: A Comprehensive Review," *Proceedings of the Fifth International Conference on Inventive Computation Technologies* ISBN: 978-1-7281-4685-0, 2020

[15] G Arun Prasath1, K Annapurani, "Prediction of sign language recognition based on multi-layered CNN," *Multimedia Tools and Applications*, doi: https://doi.org/10.1007/s11042-023-14548-1, 2023

[16] Masood, Sarfaraz, Adhyan Srivastava, Harish Chandra Thuwal, and Musheer Ahmad, "Real-time sign language gesture (word) recognition from video sequences using CNN and RNN," *In Intelligent Engineering Informatics*, pp. 623-632. Springer, Singapore, 2018.

[17] Badhe, Purva C., and Vaishali Kulkarni, "Indian sign language translator using gesture recognition algorithm," *IEEE International Conference on Computer Graphics, Vision and Information Security (CGVIS)* IEEE, 2015.

[18] Camgoz, Necati Cihan, Ahmet Alp Kındıroglu, and Lale Akarun. "Sign language recognition for assisting the deaf in hospitals," *International Workshop on Human Behavior Understanding. Springer*, Cham, 2016

[19] Vidalon, Jose Elias Yauri, and Jose Mario De Martino. "Continuous Sign Recognition Brazilian Sign Language in a Healthcare Setting," *Journal of Communication and Information Systems* 30.1 (2015).

[20] S. C.J. and L. A., "Signet: A Deep Learning based Indian Sign Language Recognition System," *International Conference on Communication and Signal Processing (ICCSP)*, Chennai, India, pp. 0596-0600, 2019

[21] www.irjmets.com

[22] www.dzone.com

[23] Joao Ulisses, Tiago Oliveira, Paula Maria Escuderino, Nuno Escudeiro, Fernando Macial Barbosa, "ACE assisted communication for education: Architecture to support blind & deaf communication," *2018 IEEE Global Engineering Education Conference (EDUCON),* 978-1-5386-2957-4/18, 2018

[24] www.deepai.org

[25] Shagun Katoch, Varsha Singh, Uma Shanker Tiwary, "Indian Sign Language recognition system using SURF with SVM and CNN," *Array*, doi: https://doi.org/10.1016/j.array.2022.100141, 2022

[26] www.research.spit.ac.in

[27] www.stats.library.ucf.edu

[28] Hsien-I Lin, Ming-Hsiang Hsu, and Wei-Kai Chen, "Human Hand Gesture Recognition Using a Convolution Neural Network," *IEEE International Conference on Automation Science and Engineering (CASE),* DOI: 10.1109/CoASE.2014.6899454, 2014

REPORT

by Khushboo Chandrakanta

Submission date: 25-Jun-2023 09:13PM (UTC+0530)
Submission ID: 2122282729
File name: ABSTRACT.pdf (578.72K)
Word count: 7109
Character count: 41968

REPORT

ORIGINALITY REPORT

9%	6%	3%	5%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	github.com Internet Source	1%
2	turcomat.org Internet Source	1%
3	www.irjmets.com Internet Source	<1%
4	Submitted to QA Learning Student Paper	<1%
5	Submitted to Taylor's Education Group Student Paper	<1%
6	Submitted to CSU Northridge Student Paper	<1%
7	vocal.media Internet Source	<1%
8	Shagun Katoch, Varsha Singh, Uma Shanker Tiwary. "Indian Sign Language recognition system using SURF with SVM and CNN", Array, 2022 Publication	<1%