International Journal of Scientific Research & Engineering Trends
Volume 9, Issue 3, May-Jun-2023, ISSN (Online): 2395-566X

Sign Language Facilitator

Asst. Prof Ms. Geeta N. Brijwani, Mr. Labhesh Joshi, Ms. Krasia Noronha
Department of Computer Science
KC College,Mumbai, India
geeta.brijwani@kccollege.edu.in, labhjoshi17@gmail.com, krasianoronha@gmail.com

Abstract- The authors proposed a method that utilizes a computer or laptop web camera to create a real-time sign language dataset. They employ technologies such as Keras and CNN Model to develop a sign language recognition system. This system aims to bridge the communication gap between individuals who are deaf and non-signers. The proposed system consists of four modules: image capture, pre-processing, training, and prediction.

Keywords- Python, Processing, Sign-Language, Realtime, Keras, Numpy, OpenCV, CNN, Machine Learning.

I. INTRODUCTION

Communication is an essential part of human interaction, and it is crucial that we ensure that everyone can effectively communicate with one another. Unfortunately, deaf individuals often face barriers when it comes to communication due to the language differences and lack of understanding of sign language. This is where sign language interpreters come in - they serve as a bridge between deaf individuals and those who do not know sign language.

However, the demand for sign language interpreters often outweighs the supply, resulting in longer wait times and a lack of accessibility for the deaf community. With the advancements in technology, we can now develop sign language interpreter projects that use computer vision and machine learning algorithms to interpret sign language.

In this project, we aim to develop a sign language interpreter that can recognize signs and translate them into text or speech. The interpreter will use a camera to capture the signs made by the user, and the machine learning algorithm will analyze the captured data to recognize the sign language.

II. TECHNOLOGIES USED

1. Python 3.10:
Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding; make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together.

2. Tensorflow (Keras):
TensorFlow is an open-source machine learning framework developed by Google that allows developers to create and train machine learning models. Keras is a high-level API for building and training neural networks that runs on top of TensorFlow. Keras provides a user-friendly interface for building and training deep learning models, making it easier for developers to get started with machine learning. It supports a wide range of neural network architectures, including convolutional neural networks (CNNs), recurrent neural networks (RNNs), and more.

With Keras, you can define a neural network model using a few lines of code. TensorFlow with Keras is a powerful and flexible tool for building and training deep learning models, and it is widely used in industry and academia for a wide range of machine learning applications.

3. Numpy:
NumPy provides a range of functions for manipulating arrays, such as reshaping, slicing, and concatenating arrays, as well as functions for performing mathematical operations on arrays, such as adding, subtracting, multiplying, and dividing arrays. NumPy can be used to process and analyze image data from video streams. For example, you could use NumPy to convert the images from the video stream into arrays of pixel values, which can then be used as input to a machine learning model.

4. Open-Source Computer Vision:
OpenCV (Open-Source Computer Vision) is a popular open-source library for computer vision and image processing tasks. It provides a wide range of functions and algorithms for image and video processing, including object detection, face recognition, and feature detection.

OpenCV can be used to process and analyze video streams to detect and recognize signs. For example, you could use OpenCV to perform background subtraction, to isolate the foreground object (i.e., the signer's hands) from the background.

OpenCV also provides a range of functions for feature detection and extraction, such as edge detection, corner detection, and blob detection. These features can be used to identify and track the movement of the signer's hands, and

to recognize specific signs based on their shape and movement.

5. HandDetector Module:
Cvzone (HandDetector Module) is an open-source computer vision library for Python, which provides a range of tools and utilities for working with images and videos. One of the key modules in cvzone is the HandDetector module, human hands in images and videos. The HandDetector module uses a convolutional neural network (CNN) to detect hands in images and videos.

6. Mediapipe:
One of the key modules in MediaPipe is the Hands module, which provides a pre-trained deep learning model for detecting and tracking hands in images and videos. The Hands module can detect the position and orientation of the signer's hands in real-time, using a deep neural network trained on a large dataset of hand images.

7. Tkinter for GUI:
Tkinter is a standard Python library that provides a simple and easy-to-use interface for building Graphical User Interfaces (GUIs) in Python.

Tkinter provides a range of useful widgets and tools for building GUIs, such as buttons, sliders, menus, and dialog boxes, which can be used to create a range of interactive and responsive applications. It also provides support for event-driven programming, allowing you to define callbacks and event handlers that respond to user input or system events.

8. pyttsx3 for Text to Voice:
pyttsx3 is a Python library that provides a simple and easy-to-use interface for converting text to speech. It is based on the Text-to-Speech (TTS) engine provided by the Microsoft Speech API, and it can be used to generate high-quality speech output in a range of languages and voices.

pyttsx3 is used to generate voice output for the detected signs or gestures, allowing users to hear a spoken description of the signs or to receive verbal feedback on their performance.

9. Python Imaging Library:
PIL (Python Imaging Library) is a Python library that provides a range of functions and tools for working with digital images. It supports a range of common image file formats, such as JPEG, PNG, GIF, and BMP.PIL can be used to crop the video stream to focus on the signer's hands or to resize the video stream to a more manageable size for processing.

PIL provides a range of image filtering and enhancement functions, such as blurring, sharpening, and contrast adjustment, which can be used to improve the quality and clarity of the video stream.PIL can be used to perform object detection and recognition tasks on the video stream, such as detecting the signer's hands or specific signs or gestures.

10. PyEnchant:
PyEnchant is a Python library that provides spell-checking and text-manipulation tools using natural language processing techniques. It is based on the Enchant library, which is a widely-used spell-checking library for many programming languages.

PyEnchant is used to check the spelling of any text input, and it can suggest corrections for any misspelled words. PyEnchant can suggest alternative words or phrases for any input text, based on context and usage patterns. PyEnchant can analyze the morphological structure of words and provide information on their inflection, declension, and conjugation patterns.

III. SCOPE

The project scope will include:

- Developing a database of sign language gestures for training the machine learning model.
- Designing and building a system that can capture sign language gestures in real-time using a camera.
- Implementing machine learning algorithms to recognize sign language gestures accurately and translate them into text or speech.
- Providing documentation and support to enable easy deployment and maintenance of the system.
- Solving accuracy problem
- We divided whole 26 different alphabets into 8 classes in which every class contains similar alphabets, which is as follows:

[y,j]

[c,o]

[g,h]

[b,d,f,I,u,v,k,r,w]

[p,q,z]

[a,e,m,n,s,t]

All the gesture labels will be assigned with a probability. The label with the highest probability will be treated to be the predicted label.

When we gesture something, the model will identify the output and classify it into a class using mathematical operation based on hand points.

For example, if it belongs to the [a,e,m,n,s,t] class, the model will further classify it into a single alphabet a or e or m or n or s or t.

- Contextual ambiguity: Sign languages often rely heavily on context to convey meaning. A sign that means one thing in one context may mean something different in another context. It can be challenging to program a system that can accurately interpret signs in context. A sign language interpreter needs to be able to recognize and adapt to these variations.

IV. MODULES

1. Image Collection:
We used vision-based, with the help of a laptop camera as the input device to collect information of the hand. We considered a vision-based approach as it was a more natural interaction between humans and computers without the use of extra devices. There are many precautions to take in this step; the hand must be in front of a clean soft background that has sufficient lighting.

2. Extracting Handpoints and plotting it:
From the image taken by the webcam we detect the hand; this is done with the help of media pipe. We then crop the appropriate portion of the hand and apply gaussian blur to the image thereafter. The filter can be easily applied using the open computer vision library also known as OpenCV. The image is then converted to a gray image using the OpenCV library. The last step is to convert the gray image to binary image using threshold and adaptive threshold methods.

The images that were collected were taken in a good background with proper lighting. However, in real-life scenarios we don't get good lighting or clean backgrounds everywhere.To overcome this problem we detected the hand from the frame using mediapipe and pinpointed different positions in the hand. We then drew and connected those points in a simple white background using open CV.

These are the pinpoints:

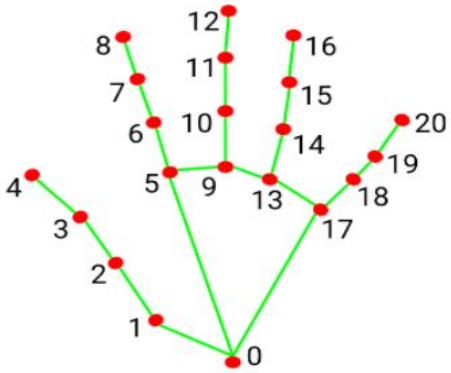

Fig 1. Pinpoints of the Hand.

3. Training Model:
Unlike regular Neural Networks, in the layers of CNN, the neurons are arranged in 3 dimensions: width, height, depth. The neurons in a layer will only be connected to a small region of the layer (window size) before it, instead of all of the neurons in a fully-connected manner. Moreover, the final output layer would have dimensions (number of classes), because by the end of the CNN architecture we will reduce the full image into a single vector of class scores.

3.1 Convolution Layer: In convolution layer we take a small window size [typically of length 5*5] that extends to the depth of the input matrix. The layer consists of learnable filters of window size. During every iteration we slid the window by stride size [typically 1], and compute the dot product of filter entries and input values at a given position. As we continue this process well create a 2-Dimensional activation matrix that gives the response of that matrix at every spatial position. That is, the network will learn filters that activate when they see some type of visual feature such as an edge of some orientation or a blotch of some color.

3.2 Fully Connected Layer: In convolution layer neurons are connected only to a local region, while in a fully connected region, well connect the all the inputs to neurons.

3.3 Final Output Layer: After getting values from fully connected layer, well connect them to final layer of neurons [having count equal to total number of classes], that will predict the probability of each image to be in different classes.

4. Text-To-Speech
We have used pyttsx3 library to convert the recognized words into the appropriate speech. The text-to-speech output is a simple workaround, but it's a useful feature because it simulates a real-life dialogue.

V. ACTIVITY DIAGRAM

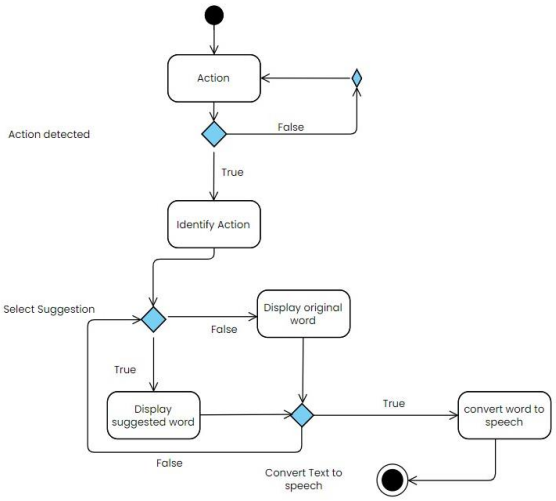

Fig 2. Activity Diagram.

The application reads the users actions. After appropriately identifying the action, the application selects a letter based on which it will suggest a word. If the suggested word is

picked, the user can click on the speak button which would convert the same word to speech.

VI. SOME HELPFUL HINTS

There are several potential enhancements that could be made to a sign language interpreter made in Python, depending on the specific goals and use cases of the project. Here are some possibilities:

1. Improved accuracy:
One of the most important enhancements would be to improve the accuracy of the interpreter. This could be achieved through better algorithms for hand tracking and gesture recognition, or by training the system on larger and more diverse datasets.

2. Real-time performance:
Another important enhancement would be to improve the real-time performance of the interpreter. This could involve optimizing the code for faster execution, using more powerful hardware, or implementing more efficient data structures.

3. Multimodal input/output:
Sign language is a complex and nuanced mode of communication that involves not only hand gestures, but also facial expressions, body language, and other forms of nonverbal communication. Enhancing the interpreter to support multimodal input and output, such as incorporating facial recognition or speech recognition, could greatly improve its overall effectiveness.

4. Integration with other applications:
Depending on the specific use case, it may be useful to integrate the sign language interpreter with other applications or systems. For example, it could be integrated with a video conferencing platform to provide real-time translation for deaf or hard-of-hearing users.

5. Support for multiple sign languages:
Sign language varies greatly across different regions and cultures. Enhancing the interpreter to support multiple sign languages could greatly increase its accessibility and usefulness for users around the world.

VII. CONCLUSION

Communication is an essential part of our daily lives, and for individuals who are deaf or hard of hearing, American Sign Language (ASL) is a vital means of communication. However, not everyone is fluent in ASL, which can create communication barriers for these individuals. To help bridge this gap, a sign language interpreter machine learning app based on ASL has been developed. This app utilizes cutting-edge machine learning algorithms and computer vision technology to accurately recognize and interpret ASL signs in real-time. With this app, individuals who are not fluent in ASL can communicate more effectively with those who are deaf or hard of hearing.

ACKNOWLEDGMENT

We would like to Ms. Geeta Brijwani for providing her constant support and guidance and inputs in the making of the project. Also, we would also like to take the opportunity to thank Mr. Aditya Agarwal and Ms. Kirti Bhatt for providing their suggestions and advice for the making of this project. Lastly, we would like to thank the college for giving us the opportunity to do research and publish a paper in a good impact research journal.

REFERENCES

[1] Aurélien Géron, O Reilly "Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow Concepts, Tools, and Techniques to Build Intelligent Systems", 2nd Edition.
[2] "Deep Learning" by Ian Goodfellow, Yoshua Bengio, and Aaron Courville
[3] "Sign Language Recognition using Machine Learning: A Survey" by Muhammad Usama, Mohammad Shahid, et al. (2019)
[4] "Sign Language Translation using Deep Learning" by Lijun Yin, Kevin W. Bowyer, and Patrick J. Flynn (2019)
[5] Tom Hope, Yehezkel S. Resheff, Itay Lieder "Learning TensorFlow: A Guide to Building Deep Learning Systems" https://www.geeksforgeeks.o rg/introduction-to-recurrent-neural-network/
[6] "Computational Models of Sign Language Processing" edited by Ronneberger, Olaf, and Sagerer, Gerhard https://www.irj et.net/archi ves/V7/I 3 /IRJET-V7I3418.pdf
[7] "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow" by Aurélien Géron
[8] Kiron Tello O'Shea, "An Introduction to Convolutional Neural Networks".
[9] "Sign Language Recognition with Convolutional Neural Networks: The Importance of Input Pose Normalization" by Gabriel de Souza et al. (2018)
[10] "American Sign Language: Linguistic and Applied Dimensions" by Ceil Lucas, Clayton Valli, and Kristin J. Mulrooney.