Go-Sign: Sign Language to Audio Application

1st Emma Brodigan
Electrical and Computer Engineering Department
Northeastern University
Boston, MA, USA
brodigan.e@northeastern.edu

2nd Matthew Downing
Electrical and Computer Engineering Department
Northeastern University
Boston, MA, USA
downing.ma@northeastern.edu

3rd Varun Khatri
Electrical and Computer Engineering Department
Northeastern University
Boston, MA, USA
khatri.va@northeastern.edu

4th Pengkai (Kyle) Lyu
Electrical and Computer Engineering Department
Northeastern University
Boston, MA, USA
lyu.pengk@northeastern.edu

5th Samruddhi Raut
Electrical and Computer Engineering Department
Northeastern University
Boston, MA, USA
raut.sa@northeastern.edu

6th Sumegha Singhania
Electrical and Computer Engineering Department
Northeastern University
Boston, MA, USA
singhania.s@northeastern.edu

***Abstract*—Sign language is the bridge between the deaf and those who can hear. Unfortunately, most hearing people do not typically know sign language, which creates a communication gap that needs to be filled. Because there are a lot of Sign Language variants from one region to another, we focused on American Sign Language (ASL), which is the predominant sign language of Deaf communities in the United States. The goal of this project was to provide an accessible web application to perform ASL translation using computer vision. This project implemented Long Short-Term Memory (LSTM) on sign language detection, which is a special type of Recurrent Neural Network. It was difficult to find a public dataset for our targeted ASL gestures, so we trained and tested the model on data we collected ourselves. It achieved an accuracy of 93% for the detection of nine different ASL gestures. We developed a web application with audio updates and visual feedback, providing a convenient and accessible way for any person with a camera to perform real-time ASL translation. Additionally, we began the development of a remote conference function, which would allow hearing people to communicate with deaf people online with real time translation.**

Keywords—American Sign Language, Long Short-Term Memory, Recurrent Neural Network, web application, assistive technology, computer vision

I. Introduction

Currently in the United States, there are about 48 million deaf or hard of hearing people. Of these, about two million people rely on American Sign Language as their main means of communication, averaging to about 22-37 million ASL interactions per day. Despite the large population of people who use ASL, there is a lack of assistive services to help integrate them into our society.

We wanted to provide an easy solution that would give them the independence and autonomy to communicate freely with the rest of society while also providing them the means to express themselves as they want to. Keeping the Principles of Universal Design in mind, we came up with an accessible, intuitive, and inexpensive solution, which was to create an app which could be used by anyone who knows how to use a simple smartphone.

II. Problem Formulation

The problem we aim to tackle with our application is the limited participation of people suffering from hearing or speech impairments in their environments, be it classrooms, offices, or recreational activities. Very often, due to a lack of efficient and adequate assistive services, such as disability trained staff in the service industry, people suffering from hearing or speech impairment do not always have the best experience. While using these services, some can feel held back from engaging with society more fully. Similarly, a lack of such services in settings like classrooms or offices can hinder their learning and intellectual growth and miss out on opportunities. It is also important for a person to be able to express themselves, as they wish to. The current assistive services such as translators may be able to efficiently relay the words, but not the emotions. Hence, we wanted to provide a solution that would help people suffering from disability to take control of their situation and empower them to interact freely with their environment, expressing themselves how they wish to.

III. Competitive Landscape

In order to reach an efficient solution, we did an analysis of the existing technologies working towards the problem elaborated previously. Currently, there is a company called Signall which is pursuing a solution towards a similar goal to ours. However, they require dedicated workstations with 3D cameras and specialized gloves which they set up at various stationary locations such as receptions, libraries, and schools. These workstations help in accurate detection of sign language and convert it into audio sentences that help speech and hearing disabled people communicate freely.

Our solution for the problem is different from Signall because it is very accessible since it's a mobile application that can be used by anyone who owns a simple smartphone. Additionally, it is also inexpensive and intuitive. It does not require any hardware, specialized equipment, or training to be utilized in daily life.

IV. Solutions Considered

The most important part of this project was gesture detection. Within gesture interaction, the key challenge is how to make hand gestures understood by computers. In conducting a literature review, we saw three main approaches for gesture detection. These approaches can be divided into three categories: “Electrical Signal Monitoring via Gloves", "BioSignal analysis based” and “Computer Vision based”.

Electrical Signal Monitoring via gloves uses an Arduino Uno board, a few flex sensors devices for digitizing hand and finger motions into multi-parametric data, and an Android application to enable communication with the users [1]. The users must wear specific gloves that connect to a system of external devices for signal processing. These devices are quite expensive and bring a much more cumbersome experience to the users.

BioSignal analysis uses an EMG (electromyographic) signal from a user’s hand and arm to control the movement of prostheses [2]. The user has electrodes placed on the surface of the skin and the signals can be measured. The measured signals are sent through signal processing via external signal processing units and the results are used to detect the hand gestures. Like Electrical Signal Monitoring via gloves, these devices are quite expensive and not very user-friendly.

Computer Vision has also been used for sign language detection by many researchers [3-7]. In contrast to the more hardware-oriented approaches, Computer Vision has the potential of carrying a wealth of information in a non-intrusive manner and at a low cost to the end user. Therefore, it constitutes a very attractive sensing modality for developing user interfaces. The Computer Vision Based methods require only a camera, thus realizing a natural interaction between humans and computers without the use of any extra devices. These systems tend to complement biological vision by describing artificial vision systems that are implemented in software and/or hardware. Though this poses a challenging problem as these systems need to be background invariant and lighting insensitive to achieve real time performance, such systems can be optimized to meet the requirements and provide great accuracy and robustness.

V. Our Solution

Our solution, Go-sign, is a web application which uses Computer Vision to detect American Sign Language and convert the hand gestures to text and audio in real time. Our aim with this project was to create an application that is accessible to the average person, without relying on any hardware besides a phone or computer, which most people already own. A mobile application is a more cost effective, accessible, and convenient solution that can be used by most people irrespective of their economic status. Instead of depending on the availability and efficiency of external accessibility features, such as trained staff or sign language interpreters, Go-sign gives the user the freedom and independence to be able to express themselves at all times and fully participate in their environments. This application is implemented using Python with OpenCV, TensorFlow, and Flask. It uses the built-in camera on the user's device to detect sign language, translates the sign to text using machine learning, and finally reads the interpretation aloud. Due to the audio aspect, this application can even be used for a hearing-impaired person to communicate with a vision-impaired person.

VI. Software Design

A. *Artificial Neural Network*

Artificial neural networks (ANNs) are computer networks that are inspired by brain biology. ANNs come in a variety of shapes and sizes. Convolutional neural networks (CNN), recurrent neural networks (RNN), and other forms of artificial neural networks (ANN) in deep learning are transforming the way humans interact with the environment. These various neural networks are at the heart of the deep learning revolution, powering applications such as unmanned aerial vehicles, self-driving automobiles, and voice recognition.

We chose an LSTM (Long Short Term Memory) artificial recurrent neural network (RNN) for this project since we are not utilizing pictures to train our model. A visual diagram of a LSTM can be seen in Figure 1. Instead, we are using an array of text that is processed using an RNN network, not a CNN network. There are other RNN networks, but we chose LSTM since it has a feedback network, whereas others only have a feed-forward network. When compared to other Artificial Neural Networks, feedback networks assist LSTM in remembering information for lengthy periods of time.

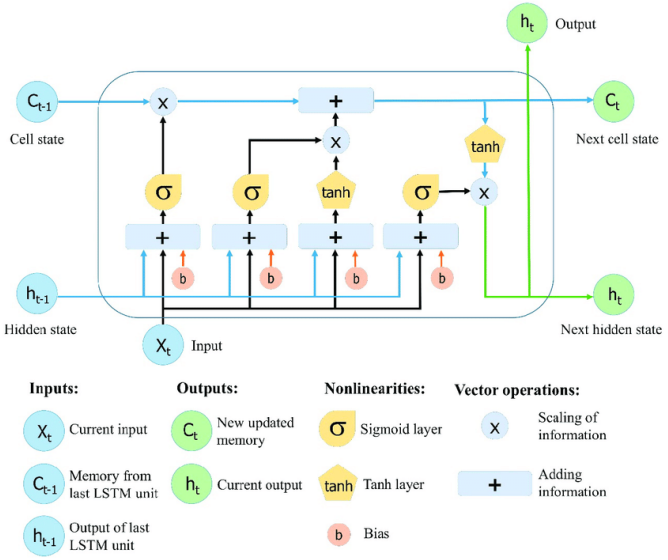

Fig. 1: Long Short Term Memory (LSTM) Artificial Recurrent Neural Network

A typical LSTM unit consists of a cell, an input gate, an output gate, and a forget gate. The cell remembers values across arbitrary time periods, and the three gates control the flow of information into and out of the cell. LSTM networks are well-suited to categorizing, analyzing, and predicting time series data.

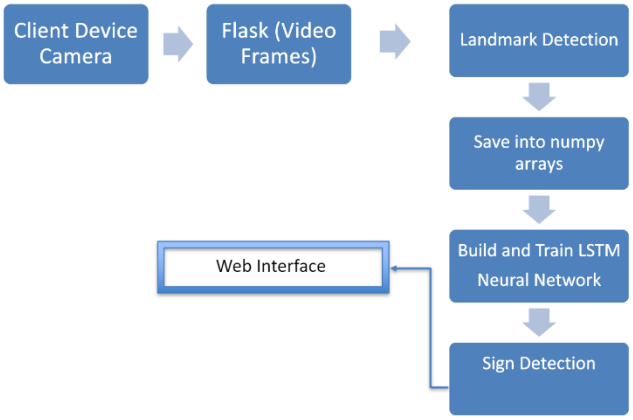

Fig. 2: Computer Vision Software Flow Chart

The Go-Sign web application is the front end of this project, and the computer vision is the back end. A visual of the flow of this can be seen in Figure 2. When a web application runs, it turns on the front/web camera which captures a live video stream and then this live video is processed using the OpenCV library to collects frames from the video. It creates a set of 30 frames, which is then used for detecting hands in each frame using the Mediapipe library [8]. A screenshot of this can be seen in Figure 3. This library also provides coordinates of twenty-one landmarks on the hand. This is then stored as arrays using the Numpy library. We use these arrays as our data set for detecting in our RNN LSTM model which is created using TensorFlow and trained on such coordinates of 21 landmarks as arrays. When our RNN model detects the Sign performed in the set of 30 frames provided by OpenCV, it provides the output results in text which is displayed on the Go-Sign web application with the photo of the sign gesture performed. This text is then converted into audio on the Go-Sign web application.

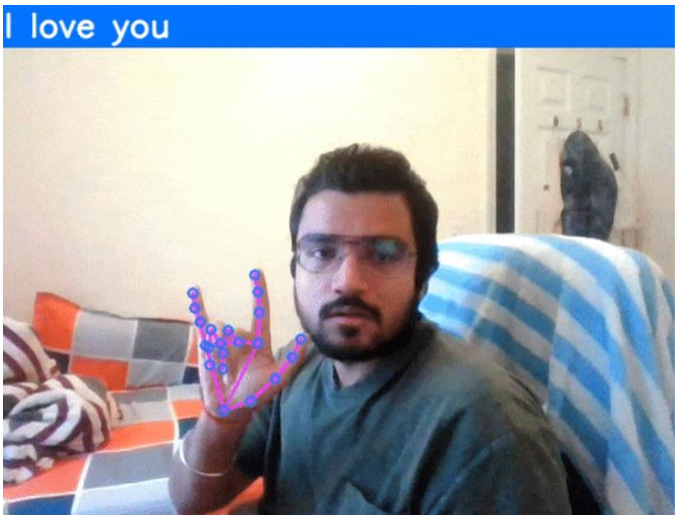

Fig. 3: Image of MediaPipe landmark detection on the right hand

B. *Data Collection*

We initially searched for a large publicly available dataset for ASL for ease of implementation and to have enough data to provide a high accuracy. However, after some online investigation, most of the public data found was using the ASL alphabet, which is not useful for translating words, which is our use case. Due to this, we decided to collect the data ourselves.

Fig. 4: The 9 ASL gestures used in this project

We collected data from six people. Each person ran through thirty times for each sign of the nine shown in Figure 4. We collected the data during many sessions. Currently, 1,800 total data items were collected for each sign. Totally, 16,200 data items have been gathered for the nine gestures. However, the size of the dataset only provided a moderately accurate model.

C. *Converting Text to Audio*

The web application was developed with the thoughts of many accessibility features. We realized we are working with the deaf and mute community, but we also wanted the app to be accessible for those who are blind. Because of this, we not only wanted a text and visual display but also auditory feedback for those groups of individuals. To accomplish this goal, we utilized the Google Text-to-speech API (Application Programming Interface), which is a service provided by Google Cloud [9]. Words or phrases can be run through the algorithm and then a .wav file (an audio file) can be generated as a result. This file can then be stored and loaded when the web app is initialized for all the common phrases. Due to this, the web app can easily play the sounds associated with the words, when the sign is detected. The computation heavy task of translation is already done ahead of time so the actual playback can happen in real time. There is also no need for a connection to the Google Cloud server when the app is running, which streamlines the app's deployment.

D. *Web Application Implementation*

The web application for this project was implemented with a Flask back-end and Javascript front-end. Flask [10], a micro web framework for Python, was chosen for the backend to ease integration with the sign detection algorithm, which was also written in Python. The Flask-SocketIO library [11] was used on top of Flask to provide access to low latency, bi-directional, event-based communications between the front-end (client) and back-end (server). This library was used to wrap the flask application, providing Socket.IO integration so that the server is

able to communicate with clients using the WebSocket protocol. On the client side, the Socket.IO Javascript library [12] was used to manage the other end of the connection. Both the client and server are able to communicate by emitting and receiving events.

The client-server architecture of the Go-sign webapp is summarized in Figure 5. First, the client initiates a socket connection with the server by sending a connection request. When the server receives this request event, it establishes a connection and sends back an acknowledgement to the client by emitting the 'connect' event. Once the connection is established, the client will periodically send the current frame data collected from the client device camera to the server using the 'image' event. Upon receiving a frame, the server will feed it into the sign detection algorithm. The sign that is detected is then emitted back to the client using the 'update sign' event. Upon receiving the 'update sign' event, the client takes care of updating the HTML elements, including text, images, and audio playback, to reflect the detected sign. This process repeats continuously.

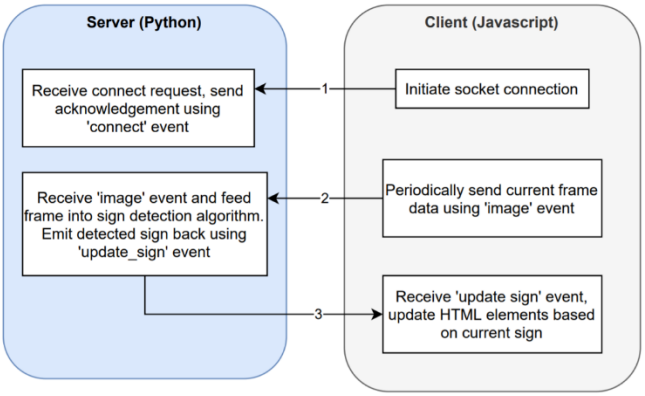

Fig. 5: Client-Server Architecture

E. Video Conferencing Feature

As an additional feature to the web application, we decided to implement sign language to audio detection in real time during video calls. For this, we built a video conferencing application using Python and Flask framework for the server-side and JavaScript, HTML and CSS for the client-side. For a customizable video conferencing experience, we utilized Twilio Programmable Video, which is a programmable cloud platform that can be used to build and customize video applications. In order to create a public URL for our video conferencing application which Twilio could connect to and run the application, we used ngrok, which is a cross-platform application that exposes local server ports to the internet. A screenshot of this application is shown in Figure 6.

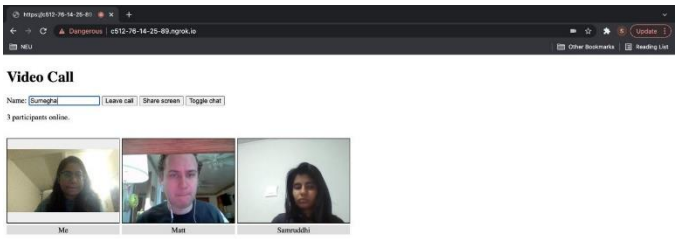

Fig 6: Video Conferencing Feature View

VII. Software Challenges

Over the course of this project, we encountered a number of challenges. Although everyone on our team had some coding experience, nobody was familiar with web development, which made many of the tasks for this project more difficult than expected. One of the first challenges we faced was determining the best method to send the updated sign data, obtained by the sign detection module, to the webpage. Initially, the solution we implemented involved updating a 'current sign' variable in the HTML template, which required the entire webpage to be refreshed at a fairly high frequency. We quickly realized that this was causing stability issues with the webpage, as well as interfering with the video feed display. To fix this, we first tried sending the updated sign information to the Javascript code using the SocketIO API. Although this was promising, we faced numerous issues getting the websocket connections to function. Another approach we tried was to have the sign detection module periodically write the current sign data to a local file. This file would then be periodically read by the Javascript code and used to update the website to reflect the current sign. Multiple problems were present with this approach. First, it would not be feasible for a scenario where this application to be actually deployed because this solution requires the sign detection module to run on the same device that the webpage is being accessed from. Second, most modern browsers do not allow webpages to access files from the local filesystem without the user explicitly selecting the file, which was not realistic for our application. Due to these reasons, we abandoned this approach and went back to debugging the issues with the websockets approach. Eventually, we were successful with this approach.

The most difficult issue we faced was integrating both the video feed and the responsive sign detecting feature into the web app. To enable camera access on any client device, we had to implement some simple Javascript code, which allows the webpage to access the local camera on whatever device the page is visited from. This became problematic because the sign detection algorithm, implemented in Python, also requires access to the camera, and most computers only allow one application at once to access the camera. To resolve this, we attempted to emit image data from the Javascript code to be passed to the python code via websockets. The image data was then fed into the sign detection algorithm to produce a result.

However, we saw that with this approach the sign detection was not functioning properly. Suspecting that the image data sent from the Javascript code was not the right format to be input in the sign detection algorithm, we tried multiple different ways of processing the data. Unfortunately, we did not have enough time and resources to solve this issue, and it remains an area for future work.

Lastly, we saw that the accuracy for sign detection dropped significantly when the algorithm was integrated into the rest of the code for the project. Unfortunately, due to time constraints, we weren't able to investigate much into this issue, so it remains an area for future work.

VIII. Final Prototype

The final prototype demonstrated the many working features in a video. A screenshot of this picture can be found in Figure 7. The web application itself consisted of four main components: a video stream to show the user what is being analyzed, the sign that has been detected in text format, the sign in a photo format, and a sound being played back through the device's speaker. The prototype relies on access to the camera of the device and a web browser, such as Google Chrome. The backend computer vision side takes in the video feed as well and uses the finger positions and movement over time to determine the appropriate sign that has been detected. This is checked against the dataset to see what resulting sign should be displayed. There were issues getting both the sign detection and video feed working together simultaneously. However, a sign update does trigger the web app to display the updated text and photo as well as begin the audio playback.

Over the last month between the midterm and final presentations, there has been significant progress. This initial prototype of three gestures (Yes, No, and Thanks) with an accuracy of 50% of the ML (machine learning) model has increased to nine signs with an ML accuracy of 93%. This can be mostly attributed to the increased data collection of many more samples including these newly added signs and increased number of people used in data collection so that the app can work more broadly. We have more fully developed and integrated the features of the web app to include the text-to-speech feature, the camera integration, and actually updating the signs and text, based on the newly detected sign.

Fig. 7: Screenshot of Web Application

IX. Future Work

In addition to the great progress that has been accomplished, there is definitely significant room for future steps. In terms of the project itself, we would like to collect more training data to further improve the accuracy of the CV algorithm. Adding more people to the dataset would also improve accuracy. In terms of the deployment of the application, we would like to convert this web app to also be used for iOS and Android. This would increase the ease of use to allow anyone to use the system on any device. Lastly, we would like to test the product on more users to get additional feedback on what is working and what additional features could be added.

In terms of additional future features, we would like to integrate the app with an audio to sign language component so that it works in both directions and can be more widely used. We would also like to implement the video conferencing feature more fully so that it can be used in video calls with multiple people. We would also like a learning mode for teaching sign language. This would be useful for when there is no active conversation but would also like to continue learning sign language. Finally, we would like to add some additional customization features to make the app feel more user friendly and unique.

X. Conclusion

Go-sign was created with the goal of assisting Hard of Hearing people by providing a web application for real-time translation of American Sign Language to text and audio. There are 2 million ASL users in the US, with an average of 22-37 million ASL interactions occurring per day. No existing solution for real-time ASL detection and translation that doesn't require the use of additional hardware was found, so our project aimed to address this. Sign language detection was accomplished by training an LSTM Neural Network with data manually generated by our team members. The neural network was implemented in Python using TensorFlow and MediaPipe, then the sign detection code was integrated into a web application composed of a Python (Flask) back-end and Javascript front-end. The sign detection module was able to detect 9 different ASL gestures, with an accuracy of 93%. The final web application we created was able to stream video from the client device camera and update the images, text, and audio playback on the webpage to reflect the sign detected by the machine learning algorithm. Many areas of future work remain, including addressing challenges with simultaneously integrating the video feed and sign detection into the webpage, integrating in the video conferencing feature we began working on, as well as improving detection accuracy and deploying to more types of devices, and more.

References

[1] Y. Jhunjhunwala, P. Shah, P. Patil, and J. Waykule, "Sign Language to Speech Conversion Using Arduino", International Journal of Recent Innovation in Engineering and Research, 2017

[2] P. Wojtczak, T. G. Amaral, O. P. Dias, A. Wolczowski, and M. Kurzynski, "Hand movement recognition based on biosignal analysis," The International Journal of Intelligent Real-Time Automation, 2008

[3] A. Halder and A. Tayade, “Real-time Vernacular Sign Language Recognition using MediaPipe and Machine Learning,” International Journal of Research Publication and Reviews, vol. 2, pp. 9-17, 2021.

[4] N. Pugeault and R. Bowden, "Spelling it out: Real-time ASL fingerspelling recognition," 2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops), 2011, pp. 1114-1119, doi: 10.1109/ICCVW.2011.6130290.

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

[6] A. Abraham, V. Rohini, “Real time conversion of sign language to speech and prediction of gestures using Artificial Neural Network,’’ 8th International Conference on Advances in Computing and Communication (ICACC-2018), 2018, vol. 143, pp. 587-594, doi: 10.1016/j.procs.2018.10.435

[7] A. Samantaray, S. K. Nayak, and A. K. Mishra, “Hand Gesture Recognition using Computer Vision,” International Journal of Scientific & Engineering Research, 2013, vol. 4, Issue 6

[8] Google, Mountain View, CA, USA, MediaPipe, ver. 0.8.9. [Online]. Available: https://google.github.io/mediapipe/

[9] P. Durette, gTTS, ver 2.2.3. [Online]. Available: https://gtts.readthedocs.io/en/latest/

[10] Pallets Projects, Flask, ver. 2.0.2. [Online]. Available: https://flask.palletsprojects.com/en/2.0.x/

[11] M. Grinberg, Flask-SocketIO, ver. 4.3.1. [Online]. Available: https://flask-socketio.readthedocs.io/en/latest/

[12] Socket.IO, ver. 2.2.0. [Online]. Available: https://socket.io/docs/v4/