Google - Isolated Sign Language Recognition

Enhance PopSign's educational games for learning ASL

With Edge Device Implementation

Nashat Cabral

Deanna Emery

Deepak Krishnamurthy

Rohan Mendiratta

Arun Surendranath

University of California, Berkeley

2023-0109 DATASCI 251 Deep Learning in the Cloud and at the Edge (251.2)

Alexandra Savelieva

Robert DesAulniers

Spring 2023

https://github.com/romendiratta/isolated-sign-language-recognition

Abstract

This project, initiated based on a Kaggle competition hosted by Google, aimed to develop a Tensorflow Lite model that can interpret American Sign Language (ASL) signs using face, hands, and pose landmark data extracted from ASL signers. The project uses machine learning and computer vision technologies to accurately identify and classify ASL signs based on the landmark data.

The model is trained on a large dataset of ASL signs and uses various techniques, such as data augmentation, transfer learning, and hyperparameter tuning, to improve accuracy. The webcam-based implementation of the model allows users to perform ASL signs in front of the camera, and the interpreted word is displayed on the screen.

The project is inspired by the need to improve accessibility and communication for individuals who rely on sign language. By providing an accurate and user-friendly means of interpreting ASL signs, this project contributes to the broader effort of making the world more inclusive and accessible for all.

Introduction

Hearing loss in children is a critical and often overlooked issue. It represents the most common birth defect in the United States, affecting approximately 12,000 infants annually, which translates to around 3 in every 1,000 babies. Furthermore, over 90% of children with permanent hearing loss have two hearing parents, making it difficult for both children and parents to learn sign language. American Sign Language (ASL) poses a significant challenge for English speakers, akin to learning Japanese. It demands considerable time and resources that many parents may not posess.

Children experiencing hearing loss often confront various obstacles in their speech and language development, social interaction, classroom learning, and mental health. Their hearing impairment may hinder language skill acquisition, peer communication, comprehension of classroom instructions, and participation in group activities. Such difficulties can lead to isolation, frustration, and low self-esteem, ultimately affecting their mental health and overall well-being. Nevertheless, several strategies and resources are available to aid these children in their learning and social development.

PopSign, a smartphone app designed for parents with deaf children, aims to make learning American Sign Language engaging and enjoyable through an interactive game format that matches ASL signs with English words. By incorporating a sign language recognizer into the game, PopSign strives to enhance the learning experience and boost the confidence of its users, allowing them to practice signing independently.

This paper focuses on developing a sign language recognizer using the Kaggle dataset for PopSign, aiming to create a more interactive and efficient learning experience for players who wish to communicate with their loved ones through sign language. Additionally, we present an application that leverages our trained sign language recognizer to predict isolated ASL signs in real time at the edge. This research seeks to contribute to the ongoing efforts to make ASL learning more accessible, engaging, and beneficial for individuals affected by hearing loss, ultimately improving their communication abilities and overall quality of life.

Background

Bohacek et al.(2022) introduced the Sign Pose-Based Transformer (SPBT), a transformer-based model specifically designed for word-level sign language recognition. The SPBT model utilizes a two-stream input representation combining both 2D and 3D pose information extracted from sign language videos, enabling the model to better understand the complex spatial and temporal features of sign language. Through extensive experimentation, the authors demonstrated that the SPBT model achieved superior recognition accuracy compared to the previous state-of-the-art methods across three benchmark datasets, including RWTH-PHOENIX-Weather 2014, WLASL1000, and SIGNUM. The proposed SPBT model represents a significant advancement in the field of sign language recognition, providing a more accurate and robust method for interpreting and understanding signed communication.

Our methodology sets itself apart from current state-of-the-art approaches in sign language recognition by incorporating an extensive vocabulary of approximately 250 sign language words. The majority of existing research focuses on a limited set of vocabulary, which may not effectively cover the wide range of expressions necessary for successful communication. We utilized MediaPipe, a cutting-edge framework that generates 543 landmarks with over 30 frames for each vocabulary word, resulting in a substantial number of data points for each sign.

In our study, we conducted thorough exploratory data analysis to identify the most critical features for sign language recognition. This process ultimately reduced the overall number of landmarks to 227, not only halving the number of data points but also streamlining the input to our model. This refinement potentially improves the model's performance and computational efficiency.

By incorporating a more extensive vocabulary and optimizing the feature selection process, our methodology aims to develop a comprehensive and robust sign language recognizer. This approach addresses the limitations of existing state-of-the-art models, enhancing the learning and communication of American Sign Language, and ultimately improving the quality of life for individuals who rely on this form of communication. Through our extensive experimentation on data preprocessing and model architecture, we strive to optimize the sign language recognizer for the benefit of those affected by hearing loss.

Significance

By integrating a sign language recognizer into PopSign, the app could become an even more effective tool for helping parents of deaf children learn ASL. This could mitigate the risk of language deprivation syndrome in deaf children and improve communication between deaf children and their parents. Furthermore, developing a TensorFlow Lite model for ASL sign classification could have broader implications for improving access to sign language and promoting better communication and understanding between the deaf and hearing communities.

Edge Device Implementation

The key difference between our project and the Kaggle competition was in the implementation of the sign detector into an edge device. This was our unique idea to add an application that can use the model's training to provide a useful output. Through the implementation of the edge device approach, we could get a user to sign live in front of the camera and get the model to interpret the sign into a useful word from ASL. Through this, we get a tangible product in which the model’s performance can be easily evaluated.

Data

Data Collection

The dataset was created with the assistance of 21 signers provided by the Deaf Professional Arts Network from various regions across the United States. These signers rely on ASL as their primary means of communication. This group is composed of individuals with various skin tones and genders.

Each signer was provided a Pixel 4a smartphone with a preinstalled app for data collection. The app would inform the signer of the required sign, chosen randomly from a 250-word vocabulary. The process for recording consisted of the signer pressing and holding a button on the screen, then signing the requested word, and finally releasing their hold on the button. Then the video of the sign is recorded with a buffer to include .5 seconds before the recording and .5 seconds after. This use case will be similar to the one presented by the PopSign app.

The app would demonstrate a video example of the desired sign, but errors to mimic the sign would still occur in these recordings. Some examples of such errors include variants of the sign based on the signer background/region, fingerspelling a sign, incorrect signing, or failing to provide a sign. Additionally, these recordings contained various extraneous movements not relevant to signing, such as the act of scratching an itch, the ending movement from the previous sign, or the beginning movement from the next sign. Moreover, at times signers would begin recordings early/late, and end recordings early/late, which potentially negatively impacts the reliability of a submitted video. Signers in these recordings may

sign with one hand over the other, or even alternate. While these scenarios present areas of potential noise in the data, the app's recognition system must be able to account for each of them.

Data Properties

The data obtained from Kaggle consisted of three parts. The first is the landmark data. Given in the form of parquet files, these are landmarks extracted from raw videos using the MediaPipe holistic model. Although the signs were originally made with raw videos, the landmark data does not contain the video at all. Instead, it consists of just the landmarks data organized in the following manner:

- *frame* - The frame number in the raw video.
- *row_id* - A unique identifier for the row.
- *type* - The type of landmark. One of ['face', 'left_hand', 'pose', 'right_hand'].
- *landmark_index* - The landmark index number. This corresponds to the index mapping for each landmark based on the MediaPipe model library.
- *[x/y/z]* - The normalized spatial coordinates of the landmark.

Below is a look at one of the landmark parquet's first 5 rows of data.

	frame	row_id	type	landmark_index	x	y	z
0	20	20-face-0	face	0	0.568502	0.368441	-0.039637
1	20	20-face-1	face	1	0.564110	0.339710	-0.065795
2	20	20-face-2	face	2	0.565730	0.350135	-0.036722
3	20	20-face-3	face	3	0.552940	0.312719	-0.046206
4	20	20-face-4	face	4	0.563474	0.330670	-0.068906

Figure 1

The Z coordinates are not considered for our project since, according to the data and instructions submitted by Kaggle, the MediaPipe model is not fully trained to predict depth. Hence we decided to drop the Z coordinates rather than include them and potentially run into issues during model training or edge implementation.

The second part of the data was a CSV file that consisted of the following:

- *path* - The path to the landmark file.
- *participant_id* - A unique identifier for the data contributor.
- *sequence_id* - A unique identifier for the landmark sequence.
- *sign* - The label for the landmark sequence.

This CSV provided a mapping between the landmark files and the corresponding sign. The participant ID and sequence ID were also provided, although they were not relevant to the final implementation of the ML model or the edge implementation.

The final part of the data from Kaggle is the 'sign_to_prediction_index_map.json'. This provided a unique ID for each of the words that were signed and presented in the landmark files.

Exploratory Data Analysis

Before proceeding with the modeling process, it is essential to standardize the number of input frames in our dataset. To achieve this, we performed an exploratory data analysis (EDA) to gain insights into the distribution of frame counts across the dataset. As depicted in Figure 2, we found that the ASL sign videos contained an average of 38 frames.

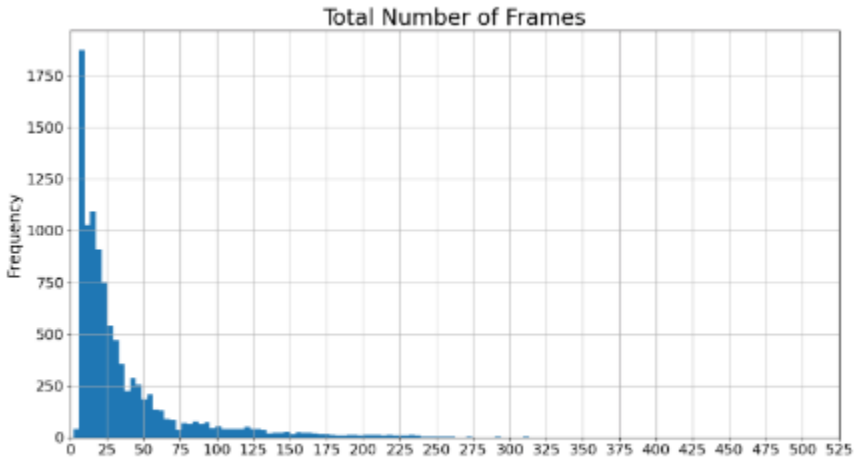


Figure 2

To provide a clearer understanding of the data, Figure 3 displays a plot of all landmarks for a single frame of a sign. As portrayed in the plot, the x and y coordinates effectively trace the person's features. In an effort o reduce the dimensionality of our data and improve the model's computational efficiency, we manually selected the individual hand, pose, and face landmarks most relevant to signing. Consequently, we only utilized the landmarks highlighted in red for our model's input. From the face, we only used landmarks for the eyes and lips, for the pose we only used the upper-body above the hips, and we used all right and left hand landmarks.

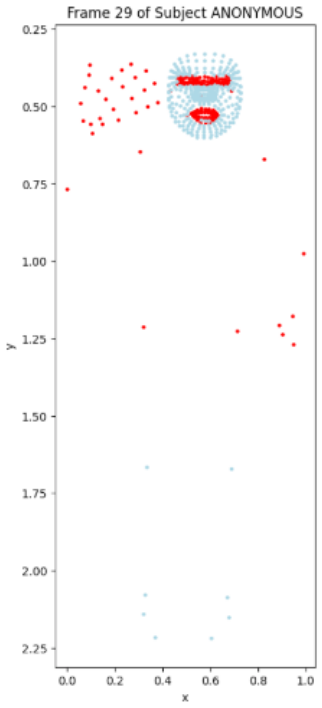


Figure 3

Lastly, examining Figure 4, which shows the count of occurrences for each sign, reveals that the classes in our dataset are well-balanced. As a result, we chose accuracy as our primary evaluation metric, aligning with the Kaggle performance metric. This choice ensures that our model's performance is accurately assessed and allows for meaningful comparisons with other models and approaches in sign language recognition.

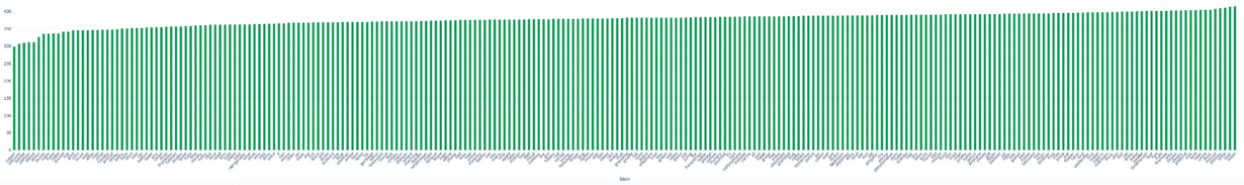


Figure 4

MediaPipe Library

MediaPipe is an open-source, cross-platform framework developed by Google for building real-time multimedia processing pipelines. MediaPipe provides pre-built models for face, pose, and hand landmark detection, which are useful for many machine learning applications involving human-computer interaction. Landmarks are specific points of interest in an image or video representing anatomical or geometric features of the subject being analyzed. In the case of human faces, pose, and hands,

landmarks represent specific points on the body parts, which provide information about the position, orientation, and movement of the body parts.

The MediaPipe Face Mesh is a solution that estimates 468 3D face landmarks in real-time, even on mobile devices[1]. A sample face mesh is shown in Figure 5.

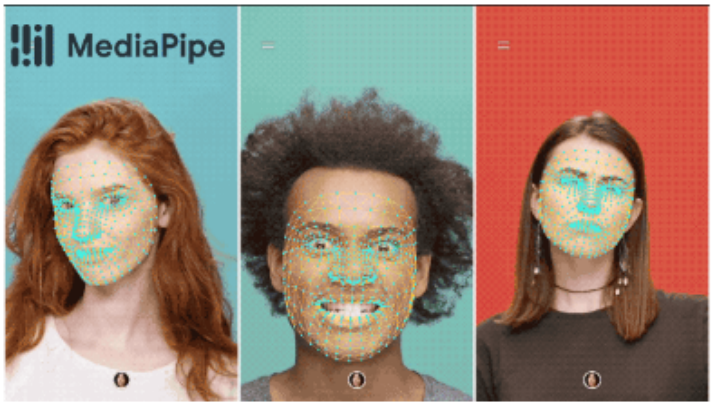
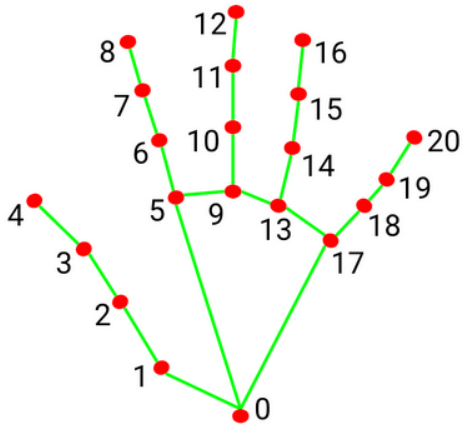


Figure 5

Source: GitHub

MediaPipe Hands is a high-fidelity hand and finger tracking solution. It employs machine learning (ML) to infer 21 3D landmarks of a hand from just a single frame [2]. Figure 6 shows the various landmarks specific to the hand.

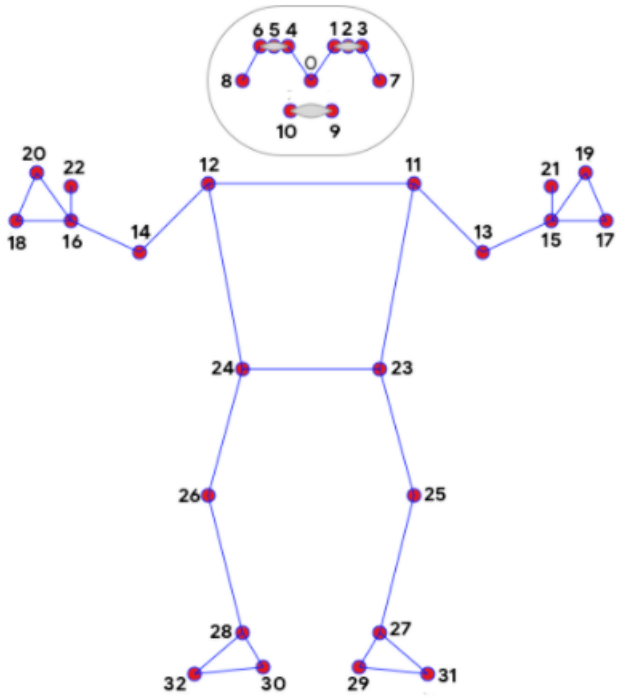


0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

Figure 6

Source: GitHub

MediaPipe Pose is an ML solution for high-fidelity body pose tracking, inferring 33 3D landmarks and background segmentation mask on the whole body.



0. nose
1. left_eye_inner
2. left_eye
3. left_eye_outer
4. right_eye_inner
5. right_eye
6. right_eye_outer
7. left_ear
8. right_ear
9. mouth_left
10. mouth_right
11. left_shoulder
12. right_shoulder
13. left_elbow
14. right_elbow
15. left_wrist
16. right_wrist
17. left_pinky
18. right_pinky
19. left_index
20. right_index
21. left_thumb
22. right_thumb
23. left_hip
24. right_hip
25. left_knee
26. right_knee
27. left_ankle
28. right_ankle
29. left_heel
30. right_heel
31. left_foot_index
32. right_foot_index

Figure 7

Source: GitHub

Methods

Approach

The approach to sign language recognition is a comprehensive, iterative process incorporating several key stages to optimize model performance. The workflow is as follows:

1. Kaggle Dataset: Utilizing a Kaggle dataset containing sign language data as the input for our model pipeline. This dataset is the foundation for training and evaluating our sign language recognition model.
2. Data Preprocessing: Once the data is acquired, preprocessing is performed to transform the raw data into a suitable format for the model. This step involves normalization, feature selection, and handling missing or noisy frames to streamline the input data, ensuring the model receives the most relevant information for accurate sign language recognition.
3. Model Development: After preprocessing the data, the development of the sign language recognition model begins. The model is designed to capture the complex spatial and temporal

features inherent in sign language data effectively.

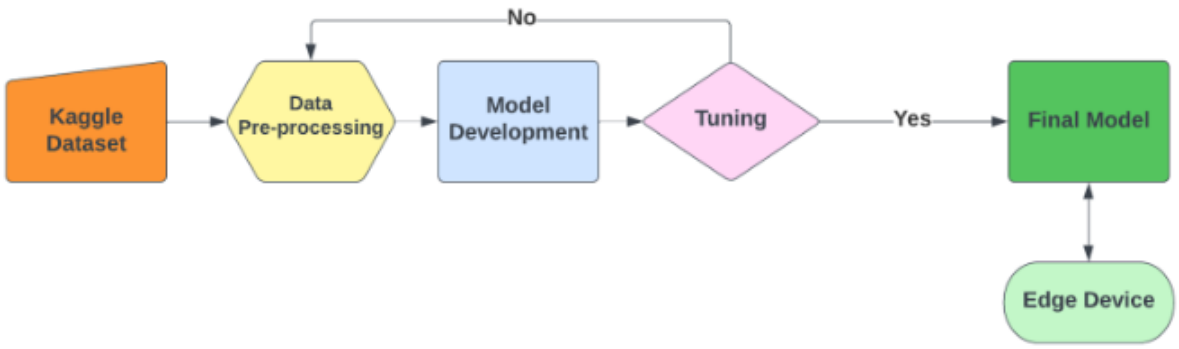


Figure 8

4. Hyperparameter Tuning: To optimize the model's performance, conduct hyperparameter tuning is conducted. This process involves experimenting with various hyperparameter settings and evaluating their impact on the model's accuracy and efficiency.
5. Feedback Loop: An essential aspect of the approach is the feedback loop created between data preprocessing, model development, and hyperparameter tuning. This iterative process refines the model based on the results obtained during tuning, leading to a more robust and accurate sign language recognizer.
6. Final Model Selection: After iterating through the feedback loop and reaching an acceptable performance level, the best-performing model is selected as the final model.
7. Edge Device Implementation: To deploy the sign language recognition model in real-world applications, the final model is converted into a TensorFlow Lite (TFLite) model. This format enables efficient model inference on edge devices, allowing users to benefit from the sign language recognizer in various settings.

Through this comprehensive, iterative approach, we aim to develop an effective and efficient sign language recognition model that can significantly improve communication for individuals affected by hearing loss. By deploying our model on edge devices, we can ensure broader accessibility and usability, ultimately enhancing the quality of life for those who rely on sign language for communication.

Data Pre-Processing

In our preprocessing layer, we implement a series of steps to ensure the input data is optimized for our sign language recognition model. These steps are detailed below:

1. Frame Removal: For each sample, we begin by dropping any frames that do not contain hand data. This step helps to eliminate noise and irrelevant information, ensuring that our model focuses on the most crucial features of sign language recognition.

2. Landmark Type Split: Next, we divide the data into four landmark types: hand, pose, face, and eyes. This separation allows us to normalize each landmark type independently, ensuring that distances between landmarks are relative and comparable.

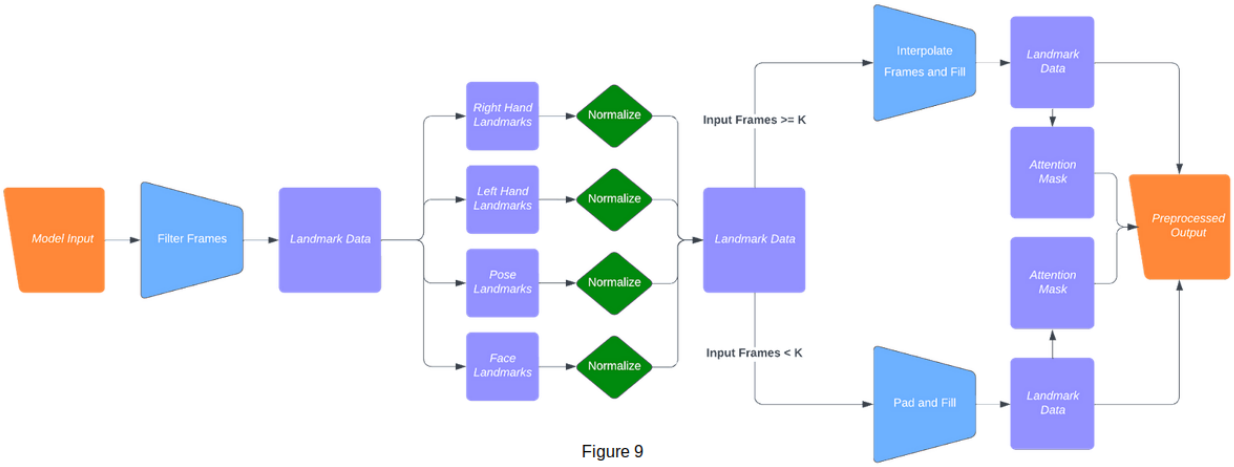


Figure 9

3. Normalization: We normalize each landmark type separately to standardize the data and ensure that the model receives consistent input, regardless of variations in the raw data. This step is crucial for improving the model's performance and generalizability.
4. Fixed Input Size Transformation: As our model requires a fixed input size, we transform the number of frames in each sample accordingly. For samples with fewer frames than the required input size, we pad them with zeros and indicate the padding in the attention mask. For samples with more frames than the input size, we interpolate frames using a nearest neighbors method to downsample the data to the desired input size.
5. Preprocessing Output: The output of our preprocessing layer consists of the processed data and its corresponding attention mask. These outputs are then used as input for our sign language recognition model, ensuring it receives well-prepared data to optimize its performance.

By implementing this comprehensive preprocessing approach, we aim to create a streamlined and consistent input for our sign language recognition model. This process helps to improve the model's performance and generalizability, ultimately resulting in a more accurate and robust sign language recognizer suitable for various applications.

Model Architecture

The high-level model architecture we propose for our sign language recognition system is designed to efficiently process raw input data from a streaming edge device and generate accurate sign predictions. The model consists of several interconnected layers and components shown in Fig. 11, each playing a vital role in the overall performance of the system. These components are described in detail below:

1. Preprocessing Component: The first layer of our model is responsible for data filtering, resizing, and normalization. This preprocessing component ensures that the raw input data is

transformed into a suitable format for the subsequent layers, addressing any inconsistencies or noise that may be present in the original data.

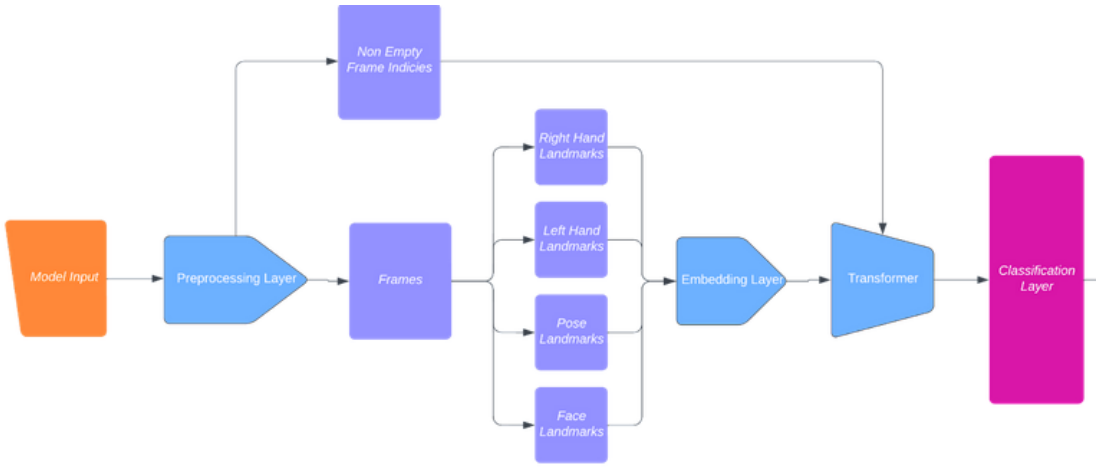


Figure 11

2. Embedding Layer: The processed frames are then sliced by landmark type and fed into an embedding layer, which efficiently encodes each landmark's information. This step allows the models to effectively capture the unique characteristics of each landmark type and understand the relationships between different landmarks.
3. Transformer Layer: The output of the embedding layer, along with an attention mask representing the non-empty frame indices, is fed into the transformer layer. This component follows a traditional transformer encoder architecture, well-suited for handling the complex spatial and temporal features in sign language data.
4. Classification Layer: The transformer output is passed through a classification layer, translating the transformed data into a sign prediction. This step allows the model to generate accurate and interpretable results, making it suitable for real-world applications.

It is important to note that all layers highlighted in blue within our model architecture are custom. This design choice enables us to perform hyperparameter tuning on specific elements within each layer, allowing for further optimization and refinement of the model's performance.

By incorporating these interconnected layers and components, our high-level model architecture aims to provide an efficient and accurate sign language recognition system capable of processing raw input data from streaming edge devices. This approach ensures that our model is well-suited for a wide range of applications, ultimately benefiting individuals who rely on sign language for communication.

Embedding Architecture

The embedding layer plays a crucial role in our sign language recognition model by efficiently encoding the input data to be used by the subsequent transformer layer. The embedding process for each landmark type is described in detail below:

1. Landmark-Specific Dense Layers: Each distinct landmark type is fed through two separate dense layers. It is important to note that each landmark type has its own set of weights, allowing the model to learn the most effective way to embed each specific type of landmark. This tailored embedding approach helps capture each landmark type's unique characteristics more accurately.
2. Stacked Output: Once the first set of dense layers processes each landmark type, their outputs are combined by stacking them together. This aggregated output preserves the information from all landmark types, ensuring that the model considers the relationships between different landmarks during the recognition process.

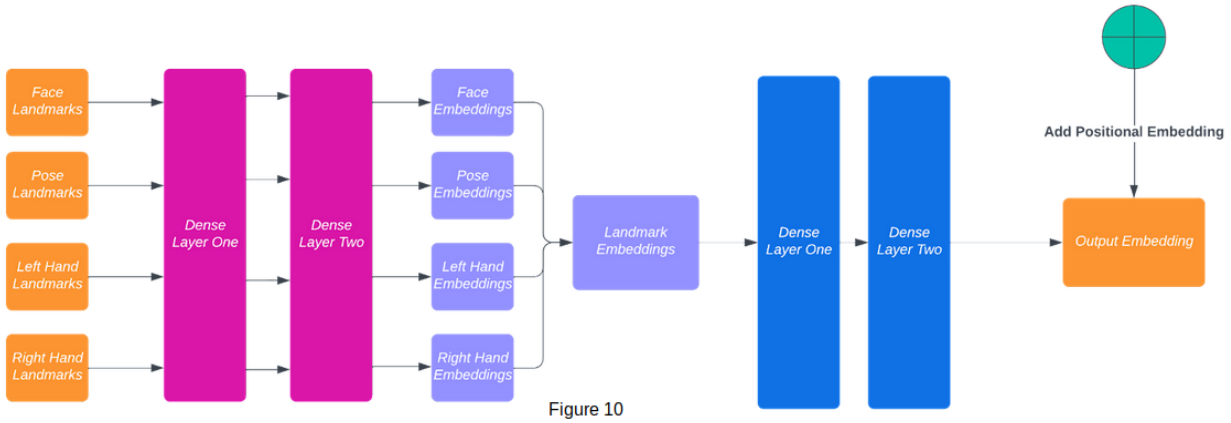


Figure 10

3. Additional Dense Layers: The stacked output is then passed through another set of dense layers, which further refines the embedding representation. This step enhances the model's ability to capture more complex patterns and relationships in the data.
4. Positional Embedding: To account for the temporal aspect of sign language data, we add a positional embedding to each frame. This step helps the model understand the order of the frames, which is essential for accurately recognizing signs that involve specific sequences or movements.
5. Transformer Layer Input: The final output of the embedding layer, which consists of the combined and refined landmark embeddings with positional information, is then fed into the transformer layer.

The transformer layer follows a traditional transformer encoder architecture, which is well-suited for handling the complex spatial and temporal features present in sign language data. Our model can efficiently encode the input data by incorporating an effective embedding layer, leading to a more accurate and robust sign language recognition model.

Edge Implementation

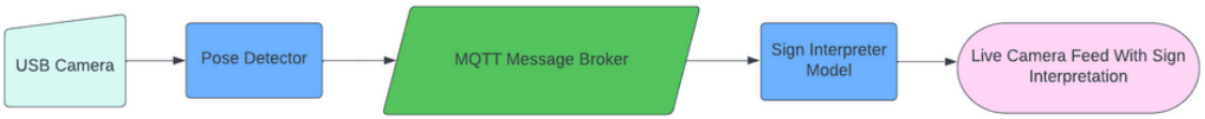
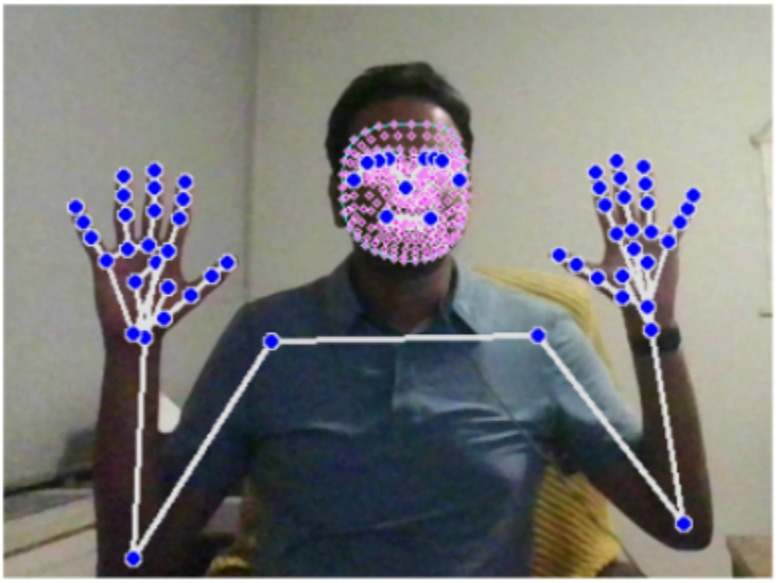


Figure 12

The Edge device architecture shown above shows the basic data flow in the edge implementation. This consisted of capturing the pose or sign through the USB camera or webcam and relaying this information to the model using a Mosquito client. The information relayed consists of the holistic landmarks data and the video frames. The model, trained on the Kaggle Isolated Sign language data, will interpret the sign from the landmarks data and output a word. This word is then displayed on screen during a live video stream.

Having Mosquito as an interface between the webcam and the model had a practical purpose. In most real-world applications, the camera is simply an interface meant to capture the frames and pass on the information for processing elsewhere. We wanted to maintain this camera functionality rather than have all the landmark processing and machine learning interpretation happening in one program. This would also allow for easier troubleshooting if required.



Detected Face, pose and Hands landmarks

Figure 13

To achieve this end, we used OpenCV to capture the signs through a live camera stream. The MediaPipe library detected the landmarks of the face, pose, and both hands. This was converted into a JSON along with the camera frames and then sent over Mosquitto to the model program. On the model's side, the data was converted back to a data frame and then processed to ensure that each frame had 543 landmarks per frame, filling in missing information with NaNs. Finally, the data was sent to the model, and the interpreted sign was displayed as text on the live stream using OpenCV.

Edge Device Use Case

The interpretation sequence for the edge device happens through the following steps once the program starts and the user is ready to sign a word. As soon as the program starts, the camera starts capturing the landmark data from the user but is not recorded until the user is ready.

1. The user presses the spacebar to begin recording.
2. The system waits for a brief buffer period of 0.5 seconds to allow the signer to get into position.
3. The prompt "Start a sign" appears on the screen to signal the user to begin signing.
4. The camera begins recording the landmarks data captured using the camera
5. The user performs their signed word, and the corresponding landmarks data is recorded into a dataframe
6. The user presses the spacebar again to conclude the recording.
7. The system drops the last half second of data from the recording to remove any noise caused by the motion of the user pressing the spacebar.
8. The model uses the recorded landmark data to generate a prediction of the signed word.
9. The predicted word is displayed on top of the live stream of the camera along with a confidence level.
10. If the prediction confidence is below 40%, this is indicated by the model.

Experiments

Throughout our research, we conducted several experiments to optimize the performance of our sign language recognition model. These experiments focused on various aspects of the preprocessing layer and the model architecture:

1. Preprocessing Layer Experiments: (a.) Normalization Techniques: We experimented with different normalization methods, such as z-score normalization, L2 normalization, and min-max scaling. These techniques aimed to standardize the input data and ensure consistent input for the model. (b.) Normalization Subsets: We also explored normalizing the data across different subsets, such as within a single sample or across all samples, to understand the impact of various normalization approaches on the model's performance. (c.) Frame Dropping: Building upon prior work, we examined the effect of dropping frames that contained no-hand landmark data. The rationale was that frames without hand data likely represented noise and could adversely affect

the model's performance. (d.) NaN Value Replacement: we tested replacing NaN values with -1 instead of 0 to indicate better that the corresponding landmarks were off-frame.

2. Model Architecture Experiments: (a.) Embedding Layer Modifications: We investigated the impact of including or excluding the embedding layer on the model's performance. This allowed us to determine the embedding layer's effectiveness in capturing each landmark type's unique characteristics. (b.) Transformer Normalization Layer: Additionally, we experimented with including or excluding the normalization layer within the transformer architecture. This exploration aimed to identify the importance of normalization in improving the model's ability to handle complex spatial and temporal features present in sign language data.

Through these extensive experiments, we sought to identify the optimal configurations and techniques for our sign language recognition model. By refining the preprocessing layer and the model architecture, we aimed to develop a more accurate, efficient, and robust system capable of recognizing a wide range of sign language expressions.

Hyperparameter Tuning

After finalizing the best-performing architecture for our model, we focused on hyperparameter tuning to further optimize its performance. Utilizing the Keras-Tuner library, specifically the Hyperband implementation, we adjusted various aspects of the model, such as learning rate, number of transformer blocks, and other model-specific parameters. The learning rate, in particular, significantly impacted model convergence, while using two transformer blocks contributed to improved accuracy. This tuning process highlights the importance of carefully selecting hyperparameters to capture complex spatial and temporal features in sign language data.

Our model's total trainable parameters ranged from 5 million to 20 million, depending on the hyperparameter combinations chosen during the tuning process. This wide range demonstrates the flexibility and scalability of our model, making it suitable for various applications and data sizes. With the tuning complete, we can confidently present our optimized sign language recognition model, designed to achieve high accuracy and efficiency in recognizing a wide range of sign language expressions.

Results

Model Performance

For final modeling, the best-performing preprocess layers and transformer architectures were selected for training and evaluation. In Figure 14 and Figure 15, the main architectural differences are described.

Tuned Model Architecture

Preprocess Layer 2 (p2)	● Filter out empty hand data frames. ● Z-score normalizes each landmark type within the frame. ● Upsample frames to input size with padding 0. ● Downsample frames to input size using nearest neighbor interpolation. ● Fill NaN values with 0.
Preprocess Layer 3 (p3)	● Filter out empty hand data frames. ● Min-max scale landmark type within the frame. ● Upsample frames to input size with 0. ● Downsample frames to input size using nearest neighbor interpolation. ● Fill NaN values with -1.

The Preprocess Layer 2 (P2) is a data preprocessing step that involves filtering out frames that do not contain any hand data. After this initial filtering, the remaining frames are Z-score normalized. This normalization process standardizes each landmark type within the frame by subtracting the mean and dividing by the standard deviation. This helps to ensure that the input data has a mean of 0 and a standard deviation of 1, which can improve the training of machine learning models. If the frames exceed the input size, they are upsampled to the desired input size with zero padding. This means that the frames are resized to the required dimensions, with any empty space filled with zeros. Else the frames are downsampled to the input size using nearest neighbor interpolation. Finally, any NaN values in the data are filled with zeros.

The Preprocess Layer 3 (P3) filters out frames that do not contain any hand data. After this, the remaining frames are min-max scaled. Like P2, the frames are upsampled to the desired input size with zero padding and then downsampled to the input size using nearest neighbor interpolation. However, in this case, any NaN values are filled with -1 instead of zeros. This is because a value of -1 can be used to

represent missing data in some machine learning algorithms, and it can help to differentiate between missing data and actual zero values in the input data.

Transformer 1 (t1)	Standard transformer encoder architecture with layer normalization removed.
Transformer 2 (t2)	Standard transformer encoder architecture.

Transformer 1 (T1) is a variant of the standard transformer encoder architecture that removes the layer normalization step. The transformer encoder is a popular neural network architecture commonly used in natural language processing (NLP) tasks. The encoder processes the input sequence frame by frame, with each frame being transformed through a series of self-attention and feedforward layers. In the standard transformer encoder architecture, layer normalization is applied after each self-attention and feedforward layer. Layer normalization is a technique that normalizes the activations of each layer to have zero mean and unit variance, helping to reduce the internal covariate shift problem and improve the stability of the training process. However, in Transformer 1, this normalization step is removed.
By removing the layer normalization step, Transformer 1 may be able to learn more complex relationships between the input frames, as the normalization step can sometimes constrain the model's ability to learn high-level representations. However, removing layer normalization can also make the training process more unstable and lead to slower convergence.

Transformer 2 (T2) is a standard transformer encoder architecture that includes layer normalization in each self-attention and feedforward layer. In this architecture, the input frames are split into multiple sub-sequences called "heads," and each head is processed in parallel through a series of self-attention and feedforward layers. After each layer, layer normalization is applied to the output of each head.

Overall, both T1 and T2 are variants of the standard transformer encoder architecture, with T1 removing the layer normalization step while T2 includes it.

Model Scores

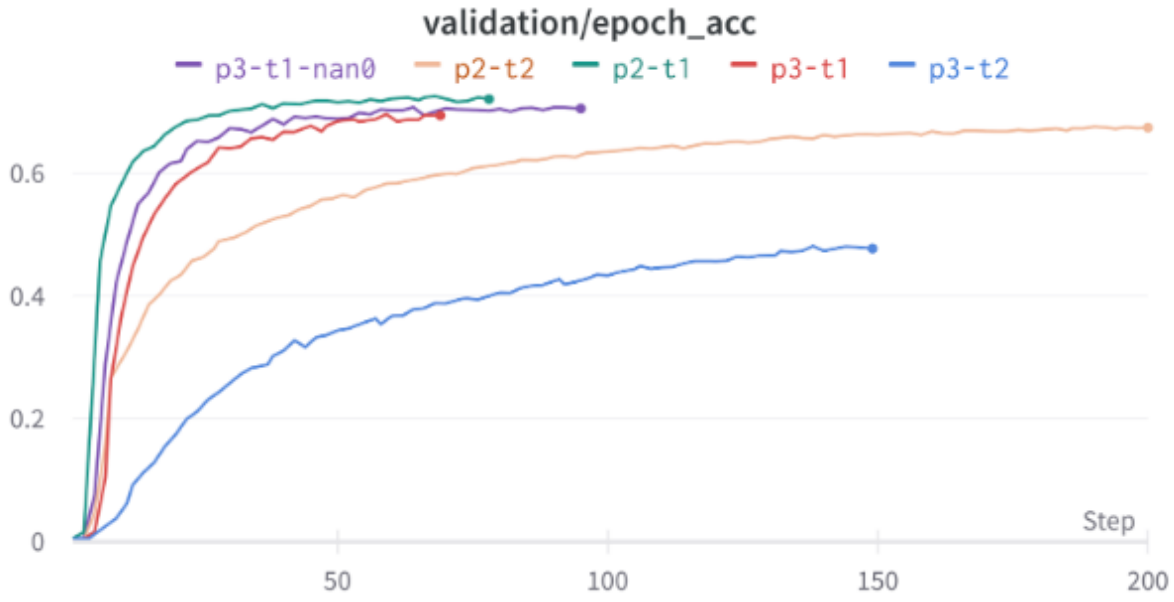


Figure 14

The performance of several preprocessing and transformer models was evaluated on a hand gesture recognition task. Figure 14 displays the validition accuracies for a few of our tests, plotted as a function of the number of steps in training. We found that Preprocess Layer 2 and Preprocess Layer 3 both performed well, with Preprocess Layer 2 outperforming Preprocess Layer 3 slightly. These preprocessing layers involved filtering out frames without any hand data, scaling and/or normalizing the remaining frames, upsampling or downsampling them to the desired input size, and filling any NaN values with zeros or -1. The use of these preprocessing steps likely improved the standardization and normalization of the input data, making it easier for the models to learn patterns and make accurate predictions.

On the other hand, Transformer 1 and Transformer 2, which are variants of the standard transformer encoder architecture, had substantially different performances. Transformer 2, which included layer normalization in each self-attention and feedforward layer, performed worse than the other models, indicating that layer normalization did not improve performance in this specific task.

Overall, the findings suggest that preprocessing steps such as filtering out irrelevant frames, scaling, and normalization can positively impact the performance of machine learning models for hand gesture recognition tasks. However, the choice of transformer architecture and hyperparameters also play important roles in determining model performance.

Further investigation is needed to determine the optimal combination of preprocessing and transformer models for this task.

Best Performing Model

Figure 15 shows the performance metrics for the *p2-t1* model, for best and worst performing sign language gestures. The results suggest that the model performs well on most sign gestures, with precision scores ranging from 0.98 to 0.38 and recall scores ranging from 0.95 to 0.32. The F1-scores range from 0.93 to 0.35, with an overall weighted average F1-score of 0.74. This model was trained with the following parameters:

- "N_EPOCHS": 150,
- "INPUT_SIZE": 38,
- "NUM_CLASSES": 250,
- "TRAIN_BATCH_SIZE": 512,
- "WD_RATIO":0.05,
- "LEARNING_RATE": 0.0001,
- "WEIGHT_DECAY": 0.0001,
- "N_WARMUP_EPOCHS": 2,
- 'LAYER_NORM_EPS' : 1e-6,
- 'LANDMARK_UNITS' : 384,
- 'TOTAL_UNITS' : 512,
- 'NUM_BLOCKS' : 2,
- 'MLP_RATIO' : 2,
- 'NUM_HEADS' : 4,
- 'MLP_DROPOUT_RATIO' : 0.10,
- 'CLASSIFIER_DROPOUT_RATIO' : 0.10

sign	precision	recall	f1-score	support	sign_ord
callonphone	0.91	0.94	0.93	33	34
for	0.98	0.89	0.93	44	88
uncle	0.91	0.95	0.93	41	228
drink	0.93	0.90	0.92	42	63
grandpa	0.97	0.88	0.92	33	100
clown	0.87	0.95	0.91	42	49
doll	0.89	0.91	0.90	34	59
tomorrow	0.92	0.88	0.90	40	221
if	0.91	0.87	0.89	60	123
icecream	0.88	0.90	0.89	41	122
mouth	0.41	0.57	0.48	40	149
wake	0.47	0.48	0.47	46	233
stay	0.54	0.42	0.47	36	205
lips	0.45	0.43	0.44	44	134
give	0.43	0.43	0.43	35	95
into	0.44	0.42	0.43	38	124
after	0.39	0.38	0.39	34	1
beside	0.38	0.32	0.35	28	21
accuracy	0.74	0.74	0.74	0	-1
macro avg	0.74	0.74	0.74	9448	-1
weighted avg	0.75	0.74	0.74	9448	-1

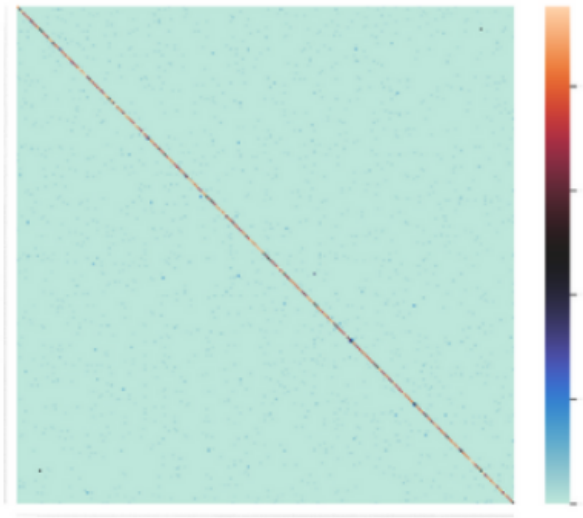


Figure 15

The model shows poor performance on some gestures, such as "mouth," "wake," "stay," and "beside," with F1-scores ranging from 0.35 to 0.48, respectively. Only 9 out of the 250 words have an F1-score under 0.5. The overall accuracy score for the model is 0.74, which is the same as the weighted average F1 score. The macro-average F1-score is also 0.74, indicating that the model performs consistently across different classes. Overall, the model shows promising results on most sign language gestures, but further improvement is required to increase the performance on the problematic gestures. Investigation into the strong diagonal in the confusion matrix provides valuable insights into the model's performance, bias, and variance. By considering the F1 scores, analysis was able to confirm further that the model has achieved a balance between precision and recall. Although minor inconsistencies were observed, the model's overall low bias and low variance indicate its ability to generalize well and perform effectively on unseen data.

Edge Device Performance

Figure 16

Performance of the edge device implementation using our sign recognition model is effective for some words and ineffective for certain other words. Some signed words such as *bed, talk, and brown* can be detected with near-perfect accuracy, and others such as *tree, blow, and elephant* can not be detected by any means. One of the reasons for this is due to limitations in the MediaPipe model. The MediaPipe model has been trained by Google on X, Y coordinates whereas the Z coordinate corresponding to depth is not available. Although the raw data provided in Kaggle has a Z coordinate available, the advice from Kaggle has been to ignore the Z data as that would not be useful in the prediction. Due to the lack of a model trained in understanding the depth coordinate, words that use depth-based signing, such as 'blow,' 'elephant', or 'tree', are harder to be perceived by the model when signed in front of the camera. This is because only a 2D equivalent sign is being passed to the model, and relevant information from the depth is not sent to the model. Consequently, it becomes difficult for the model to perceive these signs.

However, despite the potential noise in the recorded data captured in an imperfectly lit room, where the levels of contrast between skin, attire, and background may not always be ideal as well as MediaPipe's inherent limitations, this implementation shows the potential to make accurate predictions even in settings that simulate real-world conditions.

Conclusion

This paper has presented an innovative project to address the scarcity of technological resources for live ASL translation. This crucial area can significantly benefit the hearing-impaired community. With approximately 3 in 1,000 people in the United States being born with hearing impairment, developing accessible and efficient ASL translation technology is paramount.

Our project, through its submission to the Kaggle competition in an open-source format, seeks to contribute to this field and foster collaboration and innovation among researchers, developers, and practitioners. The sharing of our work in such a format allows for broader access, scrutiny, and improvement, ultimately benefiting the target community and the advancement of technology.

The continued research and development in this area have the potential to pave the way for a true "sign-to-text" application on edge devices analogous to the well-established speech-to-text features. Such an application would revolutionize communication and accessibility for the hearing impaired, enabling them to engage more seamlessly with the world around them.

Future Work

The system detailed within this report is limited to isolated single-word translation, requiring a start-stop sequence to capture and translate signed words. The team plans to expand the architecture to capture entire sentences signed by users in a live translation setting, aiming to provide more natural and fluid translations. To improve translation accuracy, the team intends to incorporate contextual prediction into the model by analyzing the context of the sentence, considering surrounding words and the grammatical structure. This approach aims to avoid mistranslations by replacing incorrectly detected words with ones that fit better in the sentence.

As the team lacks subject matter expertise in American Sign Language (ASL), they plan to collaborate with experienced ASL signers in their future work. Incorporating their feedback will be crucial in identifying and addressing any errors or inconsistencies in the system. The team also intends to work with these experts to improve the practicality of their app by incorporating their insights and feedback into edge device implementation.

Acknowledgements

We would like to express our deepest gratitude to Alexandra Savelieva and Robert DesAulniers for their invaluable guidance and instruction throughout the course, as well as their continued support during the creation and execution of our project.

References

Google. (n.d.). Google - Isolated Sign Language recognition. Kaggle. Retrieved April 20, 2023, from https://www.kaggle.com/competitions/asl-signs

Google. (n.d.). Mediapipe/docs/solutions at master · google/mediapipe. GitHub. Retrieved April 20, 2023, from https://github.com/google/mediapipe/blob/master/docs/solutions/

Markwijkhuizen. (2023, April 4). GISLR TF Data Processing & Transformer training. Kaggle. Retrieved April 20, 2023, from https://www.kaggle.com/code/markwijkhuizen/gislr-tf-data-processing-transformer-training

M. Boháček and M. Hrúz, "Sign Pose-based Transformer for Word-level Sign Language Recognition," *2022 IEEE/CVF Winter Conference on Applications of Computer Vision Workshops (WACVW)*, Waikoloa, HI, USA, 2022, pp. 182-191, doi: 10.1109/WACVW54805.2022.00024.

Nghihuynh. (2023, March 19). GISLR: EDA + feature processing. Kaggle. Retrieved April 20, 2023, from https://www.kaggle.com/code/nghihuynh/gislr-eda-feature-processing/notebook

Signs for hope. Signs for Hope | WORLDkids. (n.d.). Retrieved April 20, 2023, from https://kids.wng.org/node/6343

Staake, J. (2021, June 16). A teacher's Guide to Hearing Impairment in Children. We Are Teachers. Retrieved April 20, 2023, from https://www.weareteachers.com/children-deaf-hard-of-hearing/

World Health Organization. (n.d.). World Health Organization (WHO). World Health Organization. Retrieved April 20, 2023, from http://www.who.int/