Machine Learning and Artificial Intelligence Sign Language Detection

Pavan Kalyan, Aasish Tammana, Saurabh Rasal, Hsiao-Ping Ni

Arizona State University
Tempe, Arizona, United States - 85281

pmajjiga@asu.edu
atammana@asu.edu
srasal@asu.edu
hsiaopin@asu.edu

***Abstract*— In a world where over 70 million deaf individuals primarily communicate through sign language, the communication divide between the deaf and hearing communities remains a significant challenge. Our project addresses this challenge by developing a machine learning-based solution for real-time sign language detection and translation. Utilizing the latest advancements in computer vision and machine learning, our model is designed to interpret sign language from live video feeds, translating it into spoken or written language. By employing high-definition cameras and sophisticated image processing algorithms, our system captures and analyzes hand gestures and movements with unprecedented precision. The heart of our model is a Long Short-Term Memory (LSTM) neural network, renowned for its effectiveness in 56processing sequential data and recognizing patterns in time-series information, such as the dynamic and complex gestures of sign language.**

Keywords **- Sign Language Detection, Machine Learning, LSTM, Computer Vision, Data Preprocessing, Feature Extraction.**

I. Introduction

The realm of human communication is incredibly diverse, yet certain barriers continue to impede seamless interaction among individuals. One significant barrier is the communication gap between the deaf and hearing communities. This gap is largely due to the majority of the hearing population's unfamiliarity with sign language, the primary mode of communication for over 70 million deaf individuals worldwide.

Sign language can translate manual communication into spoken or written language, making communication more accessible for the deaf and hard-of-hearing community. A range of studies have explored the development of real-time sign language detection and translation systems through machine learning. Sonare, et al. [1] and Deep, et al. [2] emphasized the use of deep learning algorithms, such as Convolution Neural Networks (CNN) and Recurrent Neural Networks (RNN), for accurate sign language recognition. In addition, Amaya and Murray [3] used principal component analysis (PCA) and support vector machines (SVM) for real-time recognition of the five vowels in sign language. Lakhotiya [4] used CNN for American Sign Language translation. These both papers employed image classification techniques. These studies collectively highlighted the potential of machine learning for creating innovative, real-time sign language detection and translation systems.

In this project, we address this critical gap through the development of an innovative machine learning-based system for real-time sign language detection and translation. Our solution, named "Sign Language Detection", leverages the latest advancements in computer vision and machine learning to interpret sign language from live video feeds, transforming it into spoken or written language accessible to the hearing community. This endeavor aims to facilitate communication and, at the same time, foster inclusivity and empathy in our society.

II. OVERVIEW

The project's cornerstone is the integration of sophisticated computer vision techniques and a Long Short-Term Memory (LSTM) neural network. The LSTM model, known for its proficiency in processing sequential data or continuous sequences of gestures [5], is particularly suited to sign languages' dynamic and intricate nature. It is also one of the most powerful dynamic classifiers publicly known [6]. Abraham, et al. [7] and Mhatre, et al. [8] successfully implemented LSTM-based

models for sign language classification, achieving high accuracy rates. Rivera-Acosta, et al. [9] further enhanced the system by incorporating spelling correction capability, increasing model robustness. Deshpande [10] proposed the use of a larger and more diverse dataset, as well as alternative deep learning models and data preprocessing techniques, to optimize system performance. These studies collectively demonstrated the potential of LSTM-based systems for real-time sign language detection and translation.

This LSTM approach is complemented by Google's MediaPipe framework, which provides robust hand tracking and gesture recognition capabilities, essential for accurately capturing the nuances of sign language.

We also confront challenges such as data scarcity, real-time processing, the complexity of sign language, and the variability in sign language gestures, which are pivotal in achieving high accuracy in translation. Our methodology includes comprehensive data preprocessing and feature extraction to ensure that the LSTM model is trained on high-quality data, capable of recognizing a wide range of signs, including both static signs and dynamic movements.

This report details our journey in creating this transformative tool, from the initial conception to the final implementation. We discuss motivation, our objectives, the technical challenges, our innovative solutions, and the potential impact of our work.

III. Problem Statement

Sign language detection is defined as the binary-classification task for any given frame of a video if a person is using sign-language or not. So when something is being signed, the task of sign language detection is carried on [11][12]. In daily communication, around 500,000 to 2,000,000 hearing-impaired people express their thoughts through Sign Language, which is their means of communication [13]. Therefore, the motivation behind developing sign language detection systems is driven by a commitment to inclusivity, accessibility, and improved communication for individuals with hearing impairments.

The proposed sign language detection technique is hoping that the challenges mentioned earlier could be handled strategically in order to achieve certain goals. Here are our general objectives: 1) to develop a sign language dataset, ensuring representation of signs and expressions. 2) to detect real-time sign language by utilizing a computer vision and machine learning model which innovation contributes to the advancement of these fields and their application in solving real-world challenges. 3) to allow the system to adapt to different styles, speeds, and variations of signs among individual users. 4) to provide a reliable tool for effective communication between individuals with hearing impairments and the wider community. The development of a real-time sign language detection and translation system can bridge the communication gap between the deaf and hearing communities.

When it is successfully implemented, sign language detection will enable individuals with hearing impairments to communicate more effectively in a variety of settings. Furthermore, it will promote inclusivity and understanding among individuals with diverse communication needs. Finally, its applications are found in a number of different fields, including education, accessibility services, communication platforms, etc. Such systems reflect a commitment to leveraging technology to benefit society.

IV. Implementation

Our project's implementation involved a series of strategically planned steps, integrating machine learning and computer vision. Here's a overview:

1. Data Collection

We began by assembling a diverse dataset comprising sign language gestures. Given the scarcity of high-quality data, we focused on curating a dataset with a wide range of gestures, including variations in style and complexity.

2. Preprocessing

The preprocessing phase involved normalizing and scaling the data, alongside frame differencing, to prepare it for effective machine learning model training.

1) Image Resizing and Grayscale Conversion: Each image in the dataset is resized to a standard dimension and converted to grayscale. Resizing ensures uniformity in input size, while grayscale conversion simplifies the data by reducing the complexity of color channels.
2) Normalization: The pixel values of images are normalized. This standardizes the range of pixel values, making the neural network training more stable and efficient.
3) Sequence Formation for Videos: Conversion of these videos into sequences of images. This step is crucial for capturing the temporal aspects of sign language gestures.

3. Computer Vision

Computer Vision is a field of artificial intelligence that trains computers to interpret and understand the visual world. Using digital images from cameras and videos and deep learning models, computer vision systems can accurately identify and classify objects — and then react to what they "see."

The key tasks in computer vision include object recognition, image segmentation, pattern detection, and image classification. In the context of sign language detection, computer vision is used to detect and analyze hand gestures and movements.

4. Gesture Recognition with MediaPipe

MediaPipe, which is a significant component in computer vision, particularly for hand tracking and gesture recognition. MediaPipe is an open-source framework from Google that provides tools for building pipelines to perform tasks like object detection, facial recognition, and hand tracking, which are integral to your sign language detection project.

We integrated Google's MediaPipe framework for its robust hand tracking and gesture recognition capabilities. MediaPipe facilitated real-time, high-fidelity tracking of hand landmarks, enabling the detection of intricate hand movements essential for sign language interpretation. Below are the detailed set of steps involved in the implementation of Mediapipe:

1) Installing MediaPipe: We begin by installing the MediaPipe library, indicating its use for hand tracking and gesture recognition. MediaPipe's hand tracking module provides real-time, high-fidelity tracking of hand landmarks, which is crucial for detecting and interpreting sign language gestures accurately.

2) Integrating with OpenCV: Opencv-contrib-python in the dependencies entails the use of OpenCV (Open Source Computer Vision Library) alongside MediaPipe. OpenCV is widely used for tasks like image processing, video capture, and analysis. In our project, OpenCV plays a role in handling video streams and processing image data for hand gesture recognition.

3) Data Processing and Analysis: The video feed is captured and processed frame by frame. MediaPipe extracts hand landmarks from each frame, and these landmarks are then used to identify specific sign language gestures.
    - The extract_keypoints function in the code is responsible for extracting key points from the video frames. This function uses the results from MediaPipe's holistic model, which provides landmarks for pose, face, and hands (both left and right).
    - The key points for each of these parts (pose, face, left hand, right hand) are extracted as arrays of their x, y, z coordinates, and visibility (for pose landmarks). These arrays are then flattened and concatenated to form a single array representing all the key points of a frame.

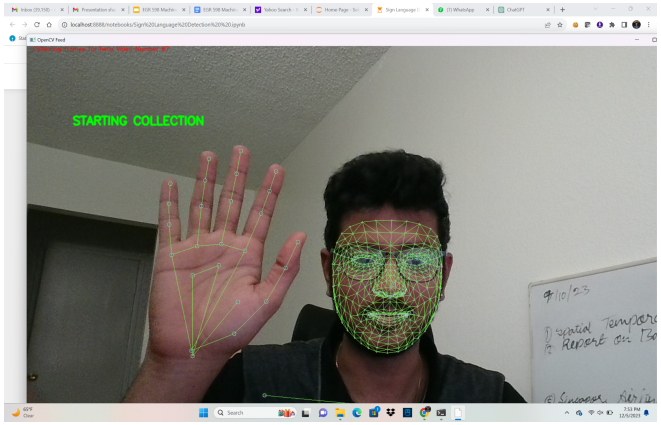

Fig. 1 Data Collection for “Hello” gesture.

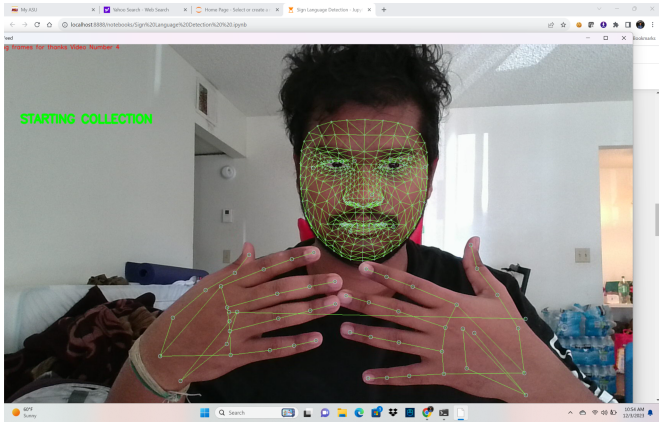

Fig. 2 Data Collection for “Thanks” gesture.

4) Gesture Recognition: The critical part of our implementation involves mapping the recognized hand landmarks to corresponding sign language gestures. This involves creating a dictionary of gestures, where each gesture is associated with a specific configuration of hand landmarks.

5) Continuous Tracking and Interpretation: In a real-time application like ours, the system continuously tracks hand movements and interprets them as they occur. This requires efficient and fast processing, ensuring that the translation of sign language to text or speech is seamless and accurate.

5. Developing the LSTM Model

At the core of our solution is a Long Short-Term Memory (LSTM) neural network. LSTM was chosen for its proficiency in handling sequential data and its ability to recognize patterns over time. This was particularly suitable for interpreting the temporal dynamics of sign language.

Long Short-Term Memory (LSTM) neural networks are a specialized form of Recurrent Neural Networks (RNNs), ideal for processing sequential data. Their unique architecture allows them to remember long-term dependencies, making them particularly effective for tasks like time-series prediction, natural language processing, and, in our case, sign language gesture recognition.

1) Model Setup

- **Libraries and Tools**: The implementation begins by importing essential modules from TensorFlow, specifically Sequential for the model architecture, LSTM and Dense for the layers, and TensorBoard for tracking the training process.
- **TensorBoard Setup**: A TensorBoard callback is configured to monitor the training process, providing a visual representation of the training metrics. This is useful for analyzing the model's performance over time.

2) Model Architecture

- **Sequential Model**: A Sequential model is chosen, which is a linear stack of layers, straightforward and well-suited for a wide range of problems.
- **LSTM Layers**:
  - The first LSTM layer has 64 units and returns sequences. This means each unit or neuron in the layer outputs a sequence of vectors instead of a single vector, which is for maintaining temporal information between layers.
  - The second LSTM layer has 128 units, also returning sequences to preserve temporal features.
  - The third LSTM layer has 64 units but does not return sequences, indicating that this is the final LSTM

layer and it should output a single vector as input to the dense layers.

- **Dense Layers**: Following the LSTM layers are Dense layers with 64 and 32 units, respectively. These are fully connected layers that learn non-linear combinations of the features extracted by the LSTM layers.
- **Output Layer**: The final Dense layer has a number of units equal to the number of actions (or classes) in the sign language dataset, using the softmax activation function for multi-class classification.

3) Model Compilation

The model is compiled with the Adam optimizer and categorical_crossentropy as the loss function, suitable for multi-class classification tasks.

The metric for evaluation is categorical_accuracy, which measures the proportion of correctly classified instances.

4) Training the Model

- The model is trained on the prepared training data (X_train and y_train) for 200 epochs.
- The TensorBoard callback is used during training, enabling the monitoring of the training process.

The LSTM model developed in our project is meticulously designed to handle the sequential nature of sign language gestures. The choice of LSTM layers with varying units and return sequences is a strategic one, ensuring the model effectively captures temporal dependencies in the data. The use of Dense layers aids in further abstracting and classifying the features extracted by the LSTM layers. The entire model is then trained over a substantial number of epochs to ensure adequate learning and generalization, with the training process monitored using TensorBoard for insights and performance improvements.

6. Real-Time Application Development

The final step involved integrating the trained model into a real-time application. This application was designed to translate sign language from video inputs into text or speech, providing a user-friendly interface for both deaf and hearing users.

1) Prediction Logic: In the real-time video capture loop, we extracted key points for each frame, appending them to a sequence. Once enough frames were collected, we fed the sequence to our trained LSTM model for prediction.
2) Visualization / Interaction: We implemented logic to visualize the predictions by displaying the detected sign language as text on the video feed. This allowed real-time interaction and verification of the model's performance.

7. Developing the Random Forest Model

Random Forest is a versatile machine learning algorithm capable of performing both regression and classification tasks. It is a type of ensemble learning method, where a group of weak models combine to form a powerful model. In the context of hand gesture recognition, a Random Forest classifier can discern patterns and classify different gestures by analyzing and learning from pre-processed image data.

1) Model Setup

- **Libraries and Tools**:Scikit-learn is utilized for constructing the Random Forest model and handling dataset splits. OpenCV supports image processing during data collection. MediaPipe extracts hand landmarks crucial for feature generation. Pickle is used for saving and loading the processed datasets and models, while NumPy provides the backbone for high-performance numerical computations.
- **Dataset Preparation**: The implementation process involves several crucial steps. Firstly, using OpenCV, we capture and collect images of various hand gestures. These images serve as the raw data for our system. Next, we perform feature extraction using MediaPipe Hands. This step is essential as it processes the captured images,

identifying and extracting hand landmarks. These landmarks serve as critical features for our Random Forest model, enabling it to recognize and classify different hand gestures effectively.

To facilitate data management and future use, we employ the Pickle library for data serialization. The extracted features, along with their corresponding labels, are serialized and stored in two separate datasets:one hand.pickle and two hand.pickle. These datasets serve as the foundation for training our Random Forest classifier, allowing us to build a robust and accurate gesture recognition system.

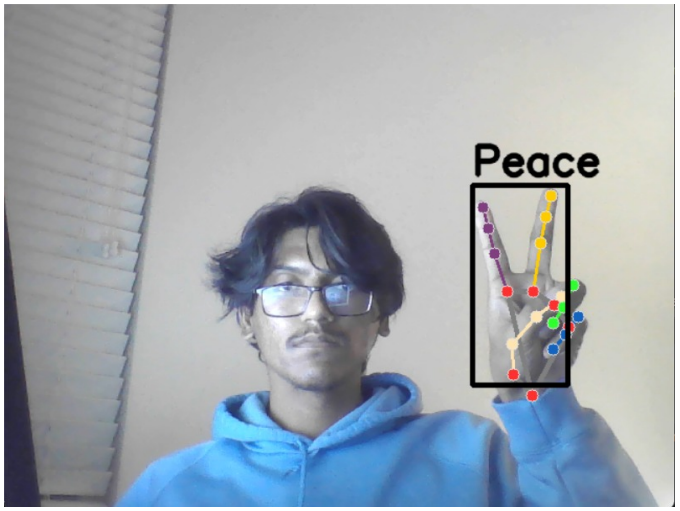

Fig. 3 Data Collection for “peace” gesture.

2) Model Architecture

- In our model architecture, we utilize two Random Forest classifiers created using Scikit-learn's RandomForestClassifier class. These classifiers are configured with default parameters, although parameter tuning for optimization is possible.
- To evaluate the model's performance, we divide each dataset into training and test sets with an 80-20 ratio. This division allows us to thoroughly assess the model's accuracy and generalization on data it hasn't seen during training, ensuring its reliability and robustness.
- During the training of each decision tree, a random subset of features (hand landmarks in our case) is selected for making split decisions. This randomness helps in reducing the correlation between trees and improving the model's generalization.
- Each decision tree in the ensemble produces a prediction. In classification tasks, the final prediction is determined by a majority vote among the individual tree predictions. This ensemble approach often results in more accurate predictions compared to a single decision tree

3) Model Compilation

Our Random Forest model architecture utilizes multiple decision trees within the ensemble, which inherently reduces overfitting without the need for external optimization. It employs a majority voting system internally for classification, and accuracy_score from Scikit-learn serves as the evaluation metric, measuring the proportion of correctly classified instances in the test data.

4) Training the Model

- The hand gesture image data undergoes preprocessing to normalize and extract relevant features. Using MediaPipe, we identify and track hand landmarks, which are crucial data points that represent the spatial position of key hand joints. These landmarks are normalized to ensure that the model is not affected by the scale and position of the hands in the images.
- Two Random Forest classifiers are trained separately: one for one-handed gestures and another for two-handed gestures. The Random Forest algorithm inherently performs feature selection during the training process, identifying the most informative features that contribute to accurate predictions.
- The ‘**fit**’ method of Scikit-learn's ‘**RandomForestClassifier**’ is used to train

the model on the training data. This process involves the construction of numerous decision trees, which are trained on different subsets of the data and feature spaces. The model learns by finding the best thresholds for splitting the data to make decisions, which are represented in the structure of the trees.

8. Evaluation and Refinement

Continuous evaluation using methods like confusion matrices helped us refine the model. We also conducted real-world testing to ensure the system's reliability and usability in diverse scenarios.

1) Model Saving: The trained model was saved for future use, enabling the deployment of the application without the need for retraining.
2) Evaluation Metrics: We employed confusion matrices and accuracy metrics to evaluate the performance of our model on test data.
3) Final Testing: The entire system was tested in real-time scenarios to ensure its functionality and performance. Adjustments were made based on these tests to optimize accuracy and usability.

V. Results

A) Model Training and Validation Loss - LSTM

During the training of our Long Short-Term Memory (LSTM) model for sign language detection, we observed the convergence of loss values over a series of epochs. The model training was conducted over 70 epochs, and both training and validation loss metrics were recorded. The attached graphical representation, titled "Model Loss," depicts a substantial decrease in loss as the number of epochs increased, signaling the model's learning progression.

Initially, a spike in training loss was noticeable, which quickly normalized as the epochs advanced. This spike is a common occurrence in training deep learning models and may be attributed to the initial adjustment of weights in response to the error gradients. As the training continued, the model's training loss showed a consistent decline, which is indicative of the model's improving ability to recognize sign language patterns within the training dataset.

Conversely, the validation loss—representing the model's performance on a separate set of data not seen during training—followed a similar downward trend, albeit with slight fluctuations. These fluctuations in validation loss suggest moments where the model's generalization to new data was not as robust, which is typical in real-world scenarios where data may vary significantly.

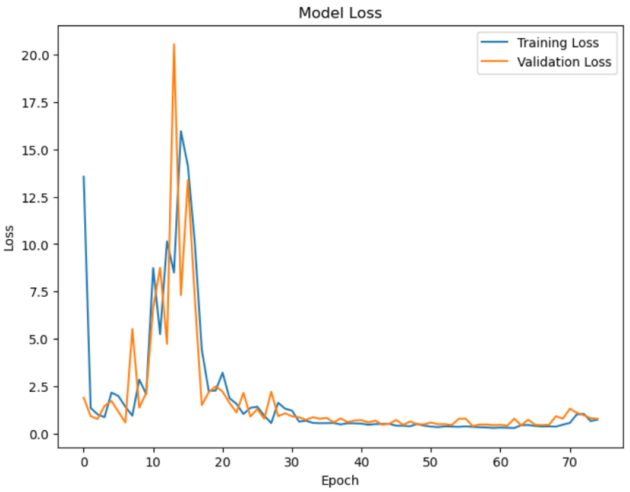

Fig. 4 Training / Validation Loss vs Number of Epochs of the Model

B) Confusion Matrix - LSTM

The confusion matrix provides a quantifiable analysis of the model's performance across different classes. In our case, the matrix evaluates the classification for three sign language gestures: "hello," "thanks," and "peace." The diagonal cells of the matrix, which represent correct predictions, are significantly populated, with "thanks" being the most accurately predicted class.

However, the off-diagonal cells indicate instances of misclassification. For example, the gesture "hello" was occasionally misclassified as "peace," and vice versa. These errors highlight potential ambiguities in the model's learning that could be related to similarities between the sign language gestures for these terms.

The results suggest that while the model has learned to distinguish between the gestures to a

considerable extent, there is room for improvement, particularly in reducing the misclassifications. Enhancing the model could involve increasing the diversity and volume of training data, introducing regularization techniques to combat overfitting, or experimenting with different model architectures and hyperparameters.

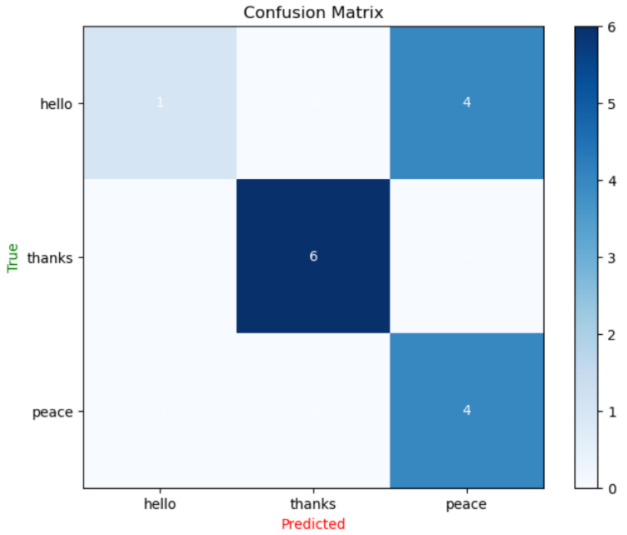

Fig. 5 Confusion Matrix for the True and Predicted classes of the Model

In conclusion, the LSTM model has demonstrated a promising ability to detect sign language gestures. Despite the challenges presented by the misclassifications, the model's performance is encouraging and lays a solid foundation for further refinement and development.

C) Confusion Matrix-Random forest classification

The confusion matrices indicate perfect classification accuracy for both Model 1 and Model 2 on the test sets, with no misclassifications observed. Model 1 correctly predicted 20 instances for two classes ('0' and '1'), while Model 2 correctly predicted 20 instances for the class labeled '0'. These results are exceptional and suggest that the models have achieved high accuracy on the provided test data.

However, it's important to note several factors:

- **Data Quality and Diversity**: We need to ensure that the test set is diverse and representative of real-world scenarios to avoid overfitting to specific data patterns.
- **Model Robustness**: We need to test the models on completely new and unseen data to validate their robustness and generalization capability.
- **Class Imbalance**: If the models need to predict more classes, we need to ensure that both the training and test sets contain a balanced representation of samples from all classes to prevent bias.
- **Evaluation Metrics**: While accuracy is essential, we need to consider other evaluation metrics such as precision, recall, and F1 score for a more comprehensive assessment, especially when dealing with an expanded number of classes.
- **Testing in Real Conditions**: Since the models performed perfectly on the test set, it's crucial to evaluate their performance in real-world conditions to ensure that they maintain high accuracy in practical applications. Real-world testing can reveal potential challenges and areas for improvement.

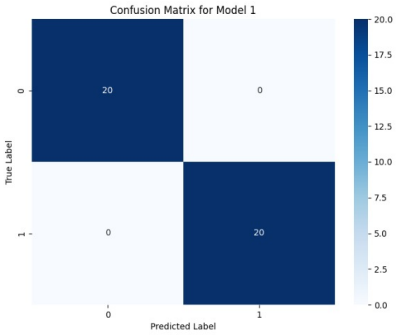

Fig. 6 Confusion Matrix for the True and Predicted classes of Model 1

(One hand gestures)

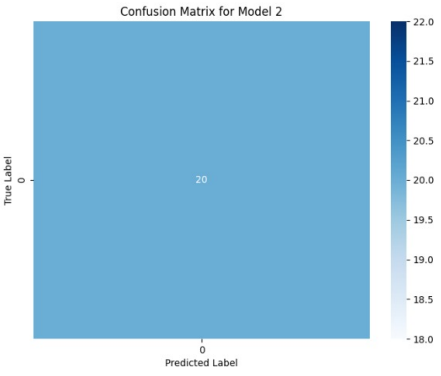

Fig. 7 Confusion Matrix for the True and Predicted classes of Model 2

(Two hand gestures)

In Conclusion the Random Forest classifiers have demonstrated their potential in accurately recognizing hand gestures. The models' abilities to classify gestures from one-handed and two-handed datasets are indicative of their robustness, supported by the high accuracy scores and clear confusion matrices. However, real-world applicability is the ultimate test, requiring the system to maintain high performance in varied and unpredictable environments.

References

[1] Sonare, B., Padgal, A., Gaikwad, Y., & Patil, A. (2021, May). Video-based sign language translation system using machine learning. In *2021 2nd International Conference for Emerging Technology (INCET)* (pp. 1-4). IEEE.

[2] Deep, A., Litoriya, A., Ingole, A., Asare, V., Bhole, S. M., & Pathak, S. (2022, August). Realtime Sign Language Detection and Recognition. In *2022 2nd Asian Conference on Innovation in Technology (ASIANCON)* (pp. 1-4). IEEE.

[3] Amaya, C., & Murray, V. (2020, September). Real-time sign language recognition. In *2020 IEEE XXVII International Conference on Electronics, Electrical Engineering and Computing (INTERCON)* (pp. 1-4). IEEE.

[4] Lakhotiya, H., Pandita, H. S., & Shankarmani, R. (2021, May). Real Time Sign Language Recognition Using Image Classification. In *2021 2nd International Conference for Emerging Technology (INCET)* (pp. 1-4). IEEE.

[5] Kothadiya, D., Bhatt, C., Sapariya, K., Patel, K., Gil-González, A. B., & Corchado, J. M. (2022). Deepsign: Sign language detection and recognition using deep learning. *Electronics*, *11*(11), 1780.

[6] Staudemeyer, R. C., & Morris, E. R. (2019). Understanding LSTM--a tutorial into long short-term memory recurrent neural networks. *arXiv preprint arXiv:1909.09586.*

[7] Abraham, E., Nayak, A., & Iqbal, A. (2019, October). Real-time translation of Indian sign language using LSTM. In *2019 global conference for advancement in technology (GCAT)* (pp. 1-5). IEEE.

[8] Mhatre, S., Joshi, S., & Kulkarni, H. B. (2022, December). Sign Language Detection using LSTM. In *2022 IEEE International Conference on Current Development in Engineering and Technology (CCET)* (pp. 1-6). IEEE.

[9] Rivera-Acosta, M., Ruiz-Varela, J. M., Ortega-Cisneros, S., Rivera, J., Parra-Michel, R., & Mejia-Alvarez, P. (2021). Spelling correction real-time american sign language alphabet translation system based on yolo network and LSTM. *Electronics*, *10*(9), 1035.

[10] Deshpande, M., Gokhale, V., Gharpure, A., Gore, A., Yadav, H., Kunekar, P., & Sawant, A. M. (2023, January). Sign Language Detection using LSTM Deep Learning Model and Media Pipe Holistic Approach. In *2023 International Conference on Artificial Intelligence and Smart Communication (AISC)* (pp. 1072-1075). IEEE.

[11] Moryossef, A., Tsochantaridis, I., Aharoni, R., Ebling, S., & Narayanan, S. (2020). Real-time sign language

[12] detection using human pose estimation. In *Computer Vision–ECCV 2020 Workshops: Glasgow, UK, August 23–28, 2020, Proceedings, Part II 16* (pp. 237-248). Springer International Publishing.

[13] Borg, M., & Camilleri, K. P. (2019, May). Sign language detection “in the wild” with recurrent neural networks. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (pp. 1637-1641). IEEE.

[14] Sharif, K., Varshini, C. S., Sreekanth, G., & Sharif, G. H. S. K. (2020). Sign Language Recognition. *International Journal of Engineering Research & Technology (IJERT). https://www. ijert. org*, *9*(5).