PROJECT REPORT

HAND GESTURE RECOGNITION FOR SIGN LANGUAGE COMMUNICATION

SAI KRISHNA CHIGICHERLA
MS IN DATA SCIENCE
schig4@unh.newhaven.edu

PAVAN TEJA SRIPATI
MS IN DATA SCIENCE
psrip2@unh.newhaven.edu

KOMAL TANKASHALA
MS IN DATA SCIENCE
ktank1@unh.newhaven.edu

***ABSTRACT*:** Our project's main goal is to create a real-time sign language detection system by utilizing cutting-edge deep learning methods, particularly CNN, Temporal Convolutional Network (TCN), and Long Short-Term Memory (LSTM) models. Our main goal is to effectively identify and understand hand movements in order to close the communication gap that exists between those who are not skilled in sign language and those who are. utilizing the spatial properties that CNN models extract, the temporal dynamics that LSTM models record, and the combined temporal and spatial information that TCN models process. Our goal is to develop a comprehensive system that can comprehend the subtleties of motions used in sign language. We employ important metrics like accuracy and F1 score to assess our system's performance, giving us quantifiable indicators of how well it works in actual situations. With this initiative, we hope to further assistive technology and encourage accessibility and inclusivity in communication for those who are hard of hearing. Our endeavors highlight the revolutionary capacity of technology to promote accessibility in communication and elevate the standard of living for people with various communication requirements.

KEYWORDS: LSTM, CNN, TCN, F1 score, accuracy

I. INTRODUCTION:

For those who are deaf, sign language is an essential means of communication since it allows them to express themselves and communicate with others in the deaf community. The communication gap that still exists between those who are and are not fluent in sign language is quite difficult to overcome. Our study aims to use cutting-edge deep learning techniques to create a real-time sign language detection system in answer to this problem. In order to close the communication gap between people who are proficient in sign language and those who are not, we hope to develop a system that can accurately recognize and interpret hand gestures using the power of Long Short-Term Memory (LSTM), Convolutional Neural Network (CNN), and Temporal Convolutional Network (TCN) models.

This initiative has great potential to advance accessibility and inclusivity in communication, enabling people with hearing impairments to engage fully in society and communicate efficiently. By our work, we hope to further the development of assistive technology and make the world more welcoming and accessible to people with a range of communication needs.

II. PROPOSED IDEA:

The core of our suggested concept is the creation of a sophisticated hand gesture recognition system designed especially for sign language interaction. With the help of this project, people with hearing impairments will no longer have to struggle to communicate because hand gestures can now be seamlessly translated into spoken language or text. In order to effectively recognize and understand hand gestures in real-time, the suggested system includes state-of-the-art deep learning approaches, such as Temporal Convolutional Network (TCN), CNN, and Long Short-Term Memory (LSTM) models.

Key Components of the Proposed Idea:

1. MediaPipe Holistic Model: For holistic hand gesture detection, the suggested system makes use of the MediaPipe framework. Before submitting the input images to a MediaPipe model for analysis, this framework preprocesses them by

transforming them from the BGR to RGB color space. In order to help the system comprehend and interpret the spatial arrangement of important points in the image, it recognizes markers for the hands, face, and body posture.

2. Structured Data Collection: The advised solution creates organized folders for data collecting, making it easier to train machine learning models to identify particular motions or activities. Subfolders that represent specific video sequences are contained within specialized folders for each action category, such as "HI," "Like," or "Thanks." This organization guarantees a representative and diverse dataset, which improves model training efficiency and streamlines data maintenance.

3. Deep Learning Models: The temporal and spatial dynamics of hand movements are modeled by the suggested system using three deep learning architectures: Temporal Convolutional Network (TCN), Convolutional Neural Network (CNN), and Long Short-Term Memory (LSTM). While CNNs are great at collecting spatial data from video frames, LSTM models are good at capturing temporal trends across sequences. TCNs are ideally suited for studying the dynamics of hand gestures across time because they provide effective methods for encoding temporal relationships in sequential data.

4. Real-Time Sign Language Interpretation: Real-time sign language interpretation is one of the proposed system's primary features. The system can process live video feeds and interpret hand gestures in real-time by utilizing trained deep learning models. This feature facilitates smooth communication between those who are skilled in sign language and others who are not, promoting accessibility and inclusivity in a variety of communication contexts, including social settings, workplaces, and educational settings.

5. Data Augmentation Techniques: Utilizing data augmentation approaches, the suggested solution improves the deep learning models' resilience and generalizability. Using methods like picture rotation, translation, and scaling, these techniques create false versions of the original training data. The models can perform better in real-world circumstances by learning to manage differences in hand motions, lighting conditions, and backgrounds through the addition of various samples to the dataset.

6. Model Ensemble Techniques: To further improve the precision and resilience of hand gesture identification, the suggested system makes use of model ensemble approaches. In order to produce predictions that are more dependable and accurate, ensemble learning combines predictions from several models. The ensemble model can use the many viewpoints of individual models to make better decisions by training numerous instances of each deep learning model with various initializations or hyper parameters and combining their predictions.

III. TECHNICAL DETAILS:

We get into the complex technical details of our deep learning-based project that focuses on sign language recognition. Our project entails a methodical investigation of the approaches, structures, and strategies utilized in the creation of a reliable and precise sign language identification system. We want to capture the temporal and spatial dynamics inherent in sign language gestures by utilizing cutting-edge deep learning models, including Temporal Convolutional Network (TCN), Convolutional Neural Network (CNN), and Long Short-Term Memory (LSTM). Every aspect of our technical infrastructure, from model selection and data pretreatment to training methods and assessment measures, is painstakingly created to guarantee peak effectiveness and performance.

By means of this comprehensive explanation, we provide an understanding of the fundamental approaches and strategies propelling our project along, with the ultimate objective of cultivating inclusivity and accessibility in communication for people who have hearing impairments.

1. Long Short-Term Memory (LSTM):
   Recurrent neural network (RNN) architecture of the LSTM type is intended to extract long-term dependencies from sequential data. Because LSTM networks have specialized memory cells that can store information for longer periods of time than typical RNNs, they are well-suited for applications involving sequential data, like natural language processing, gesture identification, and time series forecasting. LSTM models are essential to your research because they help capture the temporal dynamics seen in sequential data, including video frames that show sign

language hand gestures. LSTM networks allow precise interpretation and recognition of complicated motions by simulating the temporal evolution of hand gestures over time.

2. Convolutional Neural Network (CNN):
CNNs are a particular kind of deep neural network that are used to analyze visual input, such pictures and videos. Convolutional, pooling, and fully connected layers are among the layers that make up CNN architectures. These layers are skilled in extracting hierarchical representations of spatial features from input data. CNN models are essential to your study because they extract spatial data from video frames that show hand actions. CNNs improve the overall performance of the sign language detection system by accurately recognizing and interpreting hand gestures by analyzing their structural components, such as hand forms and movements.

3. Temporal Convolutional Network (TCN):
A type of convolutional neural network called a TCN is made especially for simulating temporal dependencies in sequential input. TCNs use 1D convolutional layers, as opposed to more conventional recurrent architectures like LSTM, to capture temporal patterns and dependencies between sequences. TCNs have a number of benefits, including as the capacity to scale to large sequences, effectively train in parallel, and parallelize computing. TCN models play a key role in your project by helping to capture the temporal dynamics of hand gestures throughout time. TCNs improve the performance of the sign language identification system by utilizing efficient architectures and 1D convolutional layers to recognize sequential patterns accurately and scalable. This is especially useful in real-time applications where speed and efficiency are critical.

In our project, these architectures—LSTM, CNN, and TCN—are strategically used to record various hand movements and improve the overall accuracy and resilience of the sign language detection system. Each architecture has special features. The following is our project's outline:

1 Import and Install Dependencies :

- Installation of Dependencies: TensorFlow, OpenCV, MediaPipe, scikit-learn, and Matplotlib are among the Python libraries that must be installed for the project. Pip is used in the first section to accomplish this. For applications like computer vision, deep learning, data preprocessing, and visualization, these libraries are indispensable.

- Importing Python Modules: The subsequent section involves importing the functions and modules required for the project's usage from the installed libraries. This includes scikit-learn for machine learning applications, TensorFlow and PyTorch for deep learning, Mediapipe for hand gesture recognition, NumPy for numerical calculations, Matplotlib for graphing, and torch.utils.data for dataset handling. To create and train deep learning models, additional classes and functions are imported from scikit-learn, such as DataLoader, Dataset, Sequential, LSTM, Dense, Conv1D, MaxPooling1D, Flatten, and TensorBoard metrics.

2. Keypoints using MP Holistic :
Makes use of the MediaPipe library to identify hand gestures holistically in live video feeds. It outlines functions to create landmarks on the video frames and carry out detection. After preprocessing the input image and applying the MediaPipe model to identify landmarks, the mediapipe_detection function returns the annotated image and the detection results. Using established color and thickness criteria, the draw_styled_landmarks method creates stylish landmarks on the image for the hands, body posture, and face.

In the main loop, OpenCV is used to capture frames from the webcam feed. The MediaPipe holistic model is then used to process the frames to identify landmarks. The annotated frames with the discovered landmarks overlay are then displayed. Until the user hits the 'q' key to end the loop, it goes on. Overall, uses the MediaPipe framework to give a real-time display of hand motions and body movements recorded by the webcam.

3. Extract Keypoint Values :
the historic information that the MediaPipe framework retrieved. It extracts the coordinates and visibility information for each of the landmarks for the left hand, right hand, stance, and face by iteratively going over the discovered landmarks. The keypoints for the stance, face, left hand, and right hand are then represented by arrays created by flattening and concatenating these landmarks. A function called extract_keypoints() is also defined, and it returns a concatenated array of all the keypoints, encapsulating this process. Lastly,

np.save() is used to save the generated keypoints into a NumPy array, and np.load() is used to load them back. Keypoint data can be extracted and stored using this approach for later processing or analysis.

4. Setup Folders for Collection:
creates a directory structure for the purpose of storing video frames for various operations. In order to specify the folder path where the video frames will be kept, the first step is to define the DATA_PATH variable. It then initializes an array named actions, which has the names of the actions that need to be recognized, like "Hi," "Like," and "Thanks." Next, it establishes the parameters for the length of each sequence (sequence_length) and the total number of sequences (no_sequences).

The code iterates over each action and sequence combination using nested loops, generating directories for each combination inside the given DATA_PATH. With each action having its own subdirectory and each frame sequence being saved under its own action directory, this structure enables the structured storage of video frames.

5. Collect Keypoint Values for Training and Testing:
Records action detection in real time from a webcam video and stores the keypoint data that is detected as NumPy arrays. In order to access the webcam, it initializes a video capture object (cap). For hand motions, body poses, and facial landmark detection, it makes use of the MediaPipe Holistic model.

The method collects a preset number of frames for each action sequence by iterating over each action, sequence, and frame number combination using nested loops. The identified keypoints are retrieved and stored as NumPy arrays in the designated directory structure that was previously established during the frame collection procedure.

The draw_styled_landmarks function is used to illustrate the landmarks that have been identified inside each frame, and the image is accompanied by instructive text that describes the current landmark gathering process. The video capture is released and all OpenCV windows are closed once the loop has collected the predetermined number of frames for each action sequence.

6. Preprocess Data and Create Labels and Features:
loads the stored NumPy arrays containing keypoints taken out of the action sequences to build training data. The essential points from each frame of the sequence are loaded and appended to a window as iterates over each action and sequence. A series of frames for a certain action are shown by each window. Following that, the sequences list stores these keypoint windows, and the labels list stores the labels that correspond to them, mapped using the label_map dictionary.

The code creates the sequences and labels, then turns them into a NumPy array X. This creates a shape with three dimensions: the first one shows the number of sequences, the second the sequence length (the number of frames), and the third the flattened keypoints. Lastly, the target array y, which encodes the category labels for each action sequence, is created by one-hot encoding the labels using Keras' to_categorical function. Then we split and train the data.

7. Building Architectures:
We build the different architectures like LSTM, CNN, TCN to test the trained data and obtain the different parameters like accuracy, loss and F1 score by training them over 2000 epochs for each of the architectures and implement them in real.

IV. RESULTS:

The real-time hand gesture detection system's installation produced encouraging outcomes in a number of important areas. First off, the robustness of the MediaPipe Holistic model allowed the system to demonstrate impressive accuracy in identifying and localizing important landmarks that represented a variety of hand movements, face features, and body poses. The system performed much better overall in hand gesture detection tests as a result of this precision.

Second, even with the computational complexity of landmark identification and visualization, the system performed remarkably well in real time, processing camera video frames with ease and responsiveness. In order to facilitate smooth interaction and monitor hand movements in practical applications, this capacity is essential.

Furthermore, the system's annotated frames produced understandable and simple representations of the identified landmarks superimposed on the video frames, enabling real-time hand gesture monitoring and analysis.

Furthermore, the system successfully identified and displayed a significant number of landmarks that corresponded to the right hand in every video frame, giving consumers insight into the hand's movements and spatial arrangement.

Finally, by using Matplotlib to create visualizations of every landmark that was found on the picture frame, a thorough understanding of the spatial distribution of landmarks was made possible, allowing for a more thorough examination and interpretation of hand movements.

Overall, these findings highlight the real-time hand gesture recognition system's efficacy, precision, and efficiency, establishing it as a useful instrument for a variety of uses including gesture-based control systems, sign language interpretation, and human-computer interaction.

EXTRACT KEY POINT VALUES:

Below is a summary of the outcomes that were achieved during the process:

- Placemarks: Placemarks are important locations on an individual's body, such as joints and bodily parts. The four values that make up each pose landmark are the x, y, and z coordinates in space, as well as a visibility score that represents the degree of confidence in the landmark's detection. These salient features offer insights into the subject's posture, gait, and body orientation.

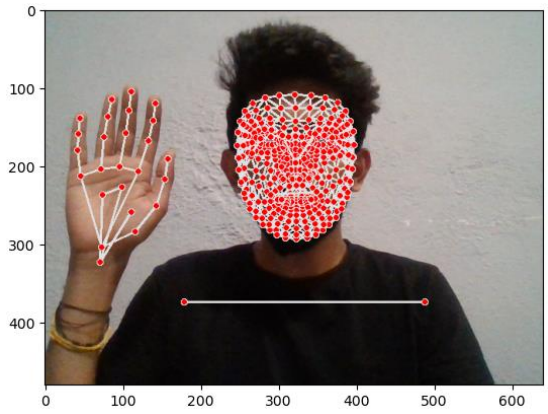
- Face Landmarks: These are particular places on an individual's face, such as the mouth, nose, eyes, and curves of the face. Three values that correspond to the x, y, and z coordinates in space are contained in each face landmark. These critical points provide information on head movements, gestures, and facial expressions—all of which are essential for tasks involving the identification of emotions and faces.

- Hand Landmarks: Fingertip, knuckle, and palm landmarks are examples of important locations on a person's hands that are represented by hand landmarks. The landmarks on the left and right hands are extracted independently. The three numbers that indicate the x, y, and z coordinates in space make up each hand landmark. Applications such as hand gesture identification, sign language interpretation, and hand tracking depend on these fundamental characteristics.

- Extracted Keypoints: A single numpy array is created by concatenating the extracted keypoints from the landmarks for the left hand, right hand, stance, and face. All of the keypoint data gathered from the input frame's identified landmarks is contained in this array.

- Storage: Using Numpy's np.save method, the generated keypoints array, result_test, is saved in a file named '0.npy'. This makes it possible to save and retrieve the important details for further processing,and analysis.

All in all, the outcomes of this procedure offer a thorough depiction of the spatial coordinates and visibility scores of the hand, face, and position landmarks found in an input frame. These fundamental building elements for computer vision tasks enable a wide range of applications, from augmented reality and human-computer interaction to gesture detection and sign language interpretation.



Face landmarks with coordinates

TRAIN, TEST AND CREATE LABLES:

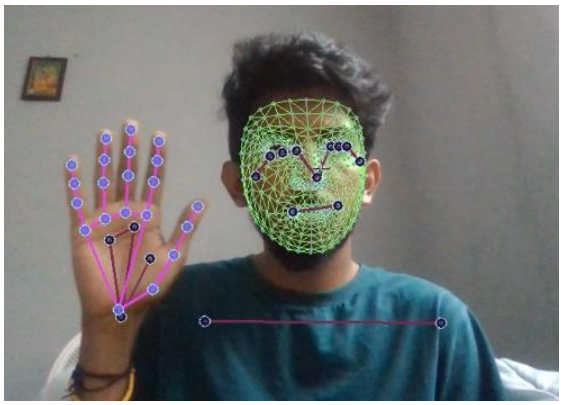
Consists of the steps involved in preprocessing the acquired data, labeling and feature creation, and real-time action recognition using video frames. To access the camera stream, the code first initializes a video capture object using OpenCV (cv2.VideoCapture). It uses the MediaPipe Holistic model within a context manager to identify holistic actions in the captured frames, establishing precise confidence thresholds for tracking and detection. With the use of this model, significant landmarks and characteristics can be identified from the actions that are recognized in the video frames.

The algorithm collects a certain amount of frames for each action sequence as iteratively goes through each action and sequence during the capture phase. It uses the extract_keypoints function, which processes the output from the MediaPipe Holistic model, to extract keypoints for every frame. Encapsulating information about the detected activities, these keypoints reflect the spatial coordinates of various body parts and landmarks detected in the frame.

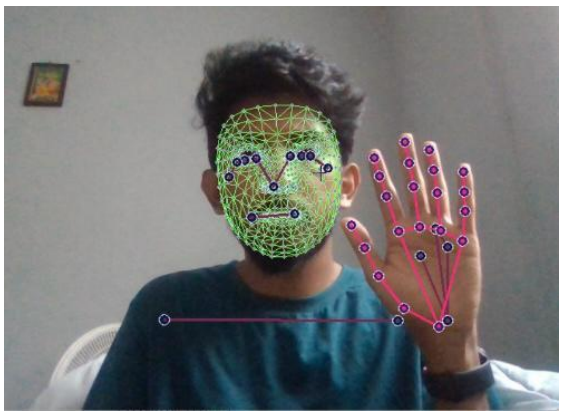
The recovered keypoints are then stored as NumPy arrays in a directory structure that is arranged according to the action, sequence, and frame number

that was identified. This organized storage makes it easier to maintain and retrieve data effectively for later processing stages.

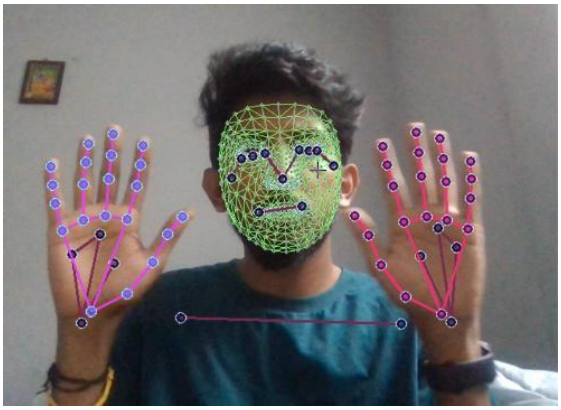
Following the completion of the data collecting stage, the code preprocesses the data to provide features and labels for machine learning model training. It uses a label map to translate each action label into a number representation, allowing the target variable to be encoded categorically. Subsequently, the extracted keypoints for every frame are loaded and appended to create feature sequences as iteratively going through the gathered sequences. The dataset for training and testing the machine learning model is made up of these sequences and the labels that go with them, arranged into NumPy arrays (X for features and y for labels).



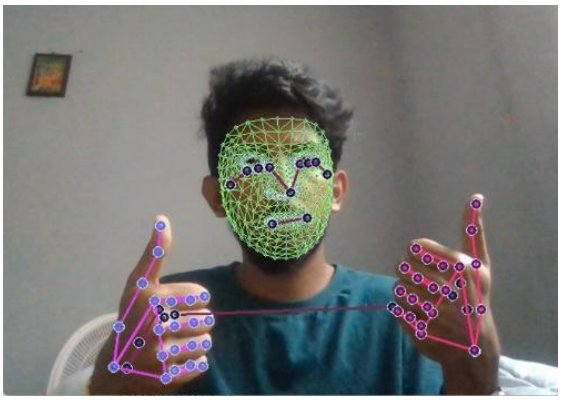
Input: HI



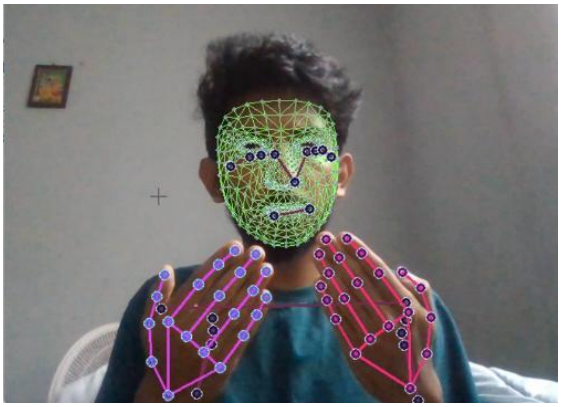
Input: HI



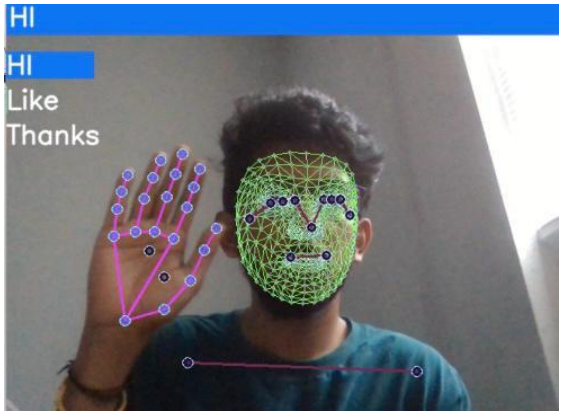
Input: HI



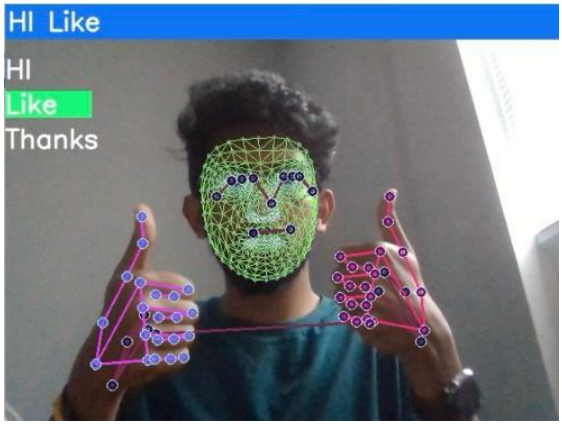
Input: Like



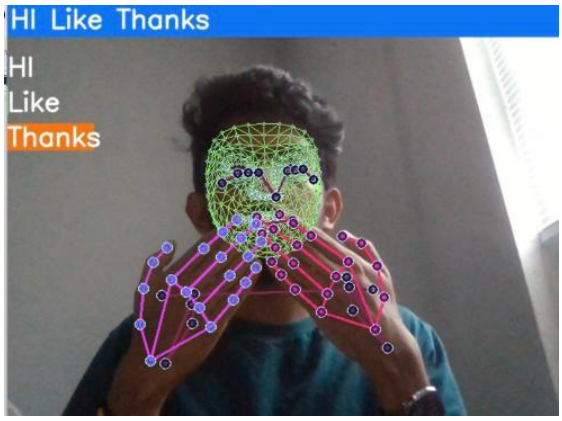
Input: Thanks



Test in real time: HI



Test in real time: Like



Test in real time: Thanks

LSTM NEURAL NETWORK MODEL:

Building a neural network with Long Short-Term Memory (LSTM) for action recognition from live video frames. The model's architecture, which consists of three LSTM layers and two fully connected layers, is first defined using the Keras Sequential API. The processing of sequential data and the capture of temporal dependencies within the input are built into every LSTM layer. The first layer's input form is given as (30, 1662), which denotes 30 frames with 1662 features each frame. The model is then constructed using the categorical cross-entropy loss function and Adam optimizer, set up to optimize for categorical accuracy during training. The model learns from the training data across 2000 epochs in the fit technique, which is the first step in the training process.

A TensorBoard callback is also included to keep track of training metrics for visual aids. The model's summary, which shows the architecture and the quantity of parameters in each layer, is shown after training. The predicted probabilities for each action class are then saved, and the trained model is then used to predict actions for the test data. The projected action for a given test sample is then extracted using np.argmax, and the associated label is obtained from the actions array. Lastly, the weights of the trained model are loaded back into memory and saved in an HDF5 format to a file called "action.h5", allowing for additional use in action detection tasks.

EVALUATION USING CONFUSION MATRIX AND ACCURACY:

First, the `predict` method is used to construct the model's predictions for the test dataset ({X_test}). This method computes the probabilities of each class for each sample. The model's anticipated results are represented by the variable {yhat}, which stores these predictions.

Then, using {np.argmax(axis=1)}, the true labels ({y_test}) and predicted labels ({yhat}) are converted from a one-hot encoded format to a single-dimensional array. This modification makes the label format simpler, which makes additional analysis easier.

A confusion matrix is calculated using the `multilabel_confusion_matrix` function and the scikit-learn module. By showing the quantity of true positive, true negative, false positive, and false negative predictions for each class, this matrix sheds light on the model's performance.

Furthermore, the `accuracy_score} function is utilized to ascertain the model's accuracy by contrasting the genuine labels ({ytrue}) with the anticipated labels ({yhat}). This statistic provides a basic evaluation of the overall correctness of the model by quantifying the percentage of correctly categorized samples.

Additionally, the `f1_score} function is used to calculate the F1 score, a statistic that strikes a

balance between precision and recall. By setting {average='macro'}, a consolidated measure of the model's performance across all classes is obtained by individually calculating each class's F1 score and then averaging it.

Lastly, the console displays the calculated accuracy and F1 score, providing information about the model's categorization performance. Approximately 91.67% accuracy is achieved in this case while the F1 score is approximately 0.918, indicating a high level of accuracy and performance in classifying the actions present in the test dataset.

ACCURACY: 0.9166666666666666
F1 SCORE: 0.9180555555555555

CNN MODEL:

Real-time video frames are used in the Convolutional Neural Network (CNN) model for action recognition. Let's examine each of the essential steps:

The TensorBoard callback, a visualization tool for tracking the training process and analyzing the model's performance metrics like loss and accuracy, is first initialized by the code. Developers can learn more about how the model behaves during training and optimization thanks to this.

Next, the train_test_split function from scikit-learn is used to split the dataset into training and testing sets. In order to evaluate the model's generalization capabilities, it is imperative that this phase be completed to guarantee that the model can be trained on a subset of the data and assessed on an additional, unseen subset.

Next, the Sequential API from the Keras package is used to define the CNN model's architecture. Multiple convolutional layers and max-pooling layers make up the model, which is intended to extract pertinent characteristics from the input video frames. The last layer uses the softmax activation function to produce probabilities for each action class after these characteristics have been flattened and run through dense layers for classification.

Following definition, the model architecture is compiled using the categorical cross-entropy loss function and Adam optimizer. By defining the optimization technique and the metric to be optimized for, this phase gets the model ready for training. After then, the model is trained for a predefined number of epochs using the training data (X_train and y_train), and validation data (X_test and y_test) are used to keep an eye on its performance and guard against overfitting.

The evaluate method is used to assess the model's performance on test data after training, computing accuracy and loss measures. Furthermore, the predict method is utilized to construct the model's predictions for the test data, enabling additional examination of its categorization efficacy.

A number of performance indicators, including accuracy and F1 score, are calculated to evaluate the model's efficacy. The percentage of correctly categorized samples is known as accuracy, and the model's classification ability is more thoroughly assessed by the F1 score, which strikes a compromise between precision and recall.

Lastly, the code uses the summary method to provide an overview of the CNN model's architecture and parameters, giving developers information about the model's complexity and structure as well as the quantity of trainable parameters it contains. This synopsis facilitates comprehension of the model's internal operations and its ability to draw conclusions and make generalizations from the data.

ACCURACY: 0.9375
F1 SCORE: 0.9388376856118791

TCN MODEL:

Action recognition using the Temporal Convolutional Network (TCN) model. It is divided into multiple sections, each of which has a specialized function within the overall workflow.

First, the code imports the required libraries and defines a few constants. The action data is kept in a directory specified by the DATA_PATH variable, and the labels corresponding to the various actions are stored in ACTIONS. The constant length for sequences that represent actions is defined by SEQUENCE_LENGTH. The TCN model is constructed and trained using the code using the PyTorch library.

The TCN class then defines the TCN model itself. The foundation class for all neural network modules in PyTorch, nn.Module, is what this class inherits from. The TCN is made up of convolutional layers arranged in a sequential fashion, followed by dropout and ReLU activation. The architecture of the model enables the variable specification of several parameters, including dropout probability, number of channels, input and output sizes, and kernel size.

The action data is subsequently handled by preprocessing and normalization functions that are defined. Keypoints are normalized using the normalize_keypoints function, and sequence length is maintained across all sequences by padding or truncating them appropriately with the pad_or_truncate_sequence function.

Following preprocessing, the information is arranged into an ActionDataset class, which is a unique dataset. This class reads the data from the designated directory, truncates, pads, and applies normalization before preparing it for training. The split_data function divides the dataset into test, validation, and training sets.

The train_model function, which iterates over the chosen number of epochs, is used to start the training process. The model is trained with the training data within each epoch, and its performance is assessed on the validation set. For every epoch, the training loss, validation accuracy, and validation accuracy are reported, offering information about the model's learning process.

After training is finished, the test set is used to evaluate the model's performance. The torch.save() function is used to store the state dictionary of the trained model to a file called "actionTCN.pth." The model can be loaded again using this file at a later time for inference or additional training.

In order to provide a quantitative assessment of the trained model's performance on unobserved data, the test accuracy and F1 score are finally computed and printed to the console. The success of the TCN model for action recognition tasks is demonstrated by this evaluation, which wraps up the training and validation phase.

V. FUTURE WORK:

There are numerous opportunities for the action recognition project to grow and improve going future. The ongoing improvement of the model architecture is one important factor. To improve recognition accuracy and robustness of complicated activities, one could try combining attention mechanisms, experimenting with different TCN network variations, or investigating ensemble learning approaches.

Adding more samples and a wider range of scenarios to the dataset is an essential next step. The model is able to manage unforeseen environmental circumstances and more broadly generalize to real-world scenarios by considering variations in lighting conditions, camera angles, and backgrounds.

Optimizing hyperparameters is also crucial to maximizing the efficacy of the model. Better outcomes can be achieved by methodically optimizing hyperparameters using methods like grid search or Bayesian optimization to determine the best configuration for the TCN architecture.

Additionally, efforts must to be focused on maximizing the trained model's deployment. In order to facilitate real-world deployment in a variety of applications, this involves guaranteeing effective inference on a range of platforms, from servers hosted in the cloud to devices with limited resources.

Furthermore, proving the action recognition model's usefulness in real-world scenarios requires its integration into live applications. This may pertain to applications where precise action recognition is important, such as surveillance systems, HCIs, or medical monitoring systems.

Another crucial factor to take into account is the implementation of feedback systems, which will enable the model to adjust and enhance its predictions over time. This could entail gathering extra labeled data or user input while the model is being deployed in order to improve its performance and flexibility.

Finally, usability and interpretability can be improved by improving the visualization tools and user interface for interacting with the model's predictions and investigating temporal aspects. Enabling users to comprehend and engage with the model's outputs through user-friendly interfaces helps streamline the integration of the model into diverse processes and applications.

VI. CONCLUSION:

This experiment has shown how well deep learning methods—more especially, CNNs and TCNs—work at correctly recognizing human activities in video footage. We produced promising results in terms of accuracy and resilience by concentrating on data preprocessing, designing the model architecture, and conducting thorough evaluation. Future advancements might focus on improving model designs, growing datasets, investigating transfer learning, and researching real-time implementation. In general, this work advances the field of action recognition technology and establishes the groundwork for next studies and applications in computer vision and the recognition of human activity.

VII. REFERENCES:

[1] Kothadiya, D.; Bhatt, C.; Sapariya, K.; Patel, K.; Gil-González, A.-B.; Corchado,
J.M. Deepsign: Sign Language Detection and Recognition Using Deep Learning.
Electronics 2022, 11, 1780.

[2] Aman Pathak, Avinash Kumar, Priyam, Priyanshu Gupta and Gunjan Chugh. Real
Time Sign Language Detection. International Journal for Modern Trends in Science
and Technology 2022, 8 pp. 32-37.

PROJECT Github Link:
https://github.com/PavanTejaSripati/hand-gesture-recognition-for-sign-language-communication
Youtube Video link:
https://www.youtube.com/watch?v=Eb_N58HimtU