Sign Language Recognition

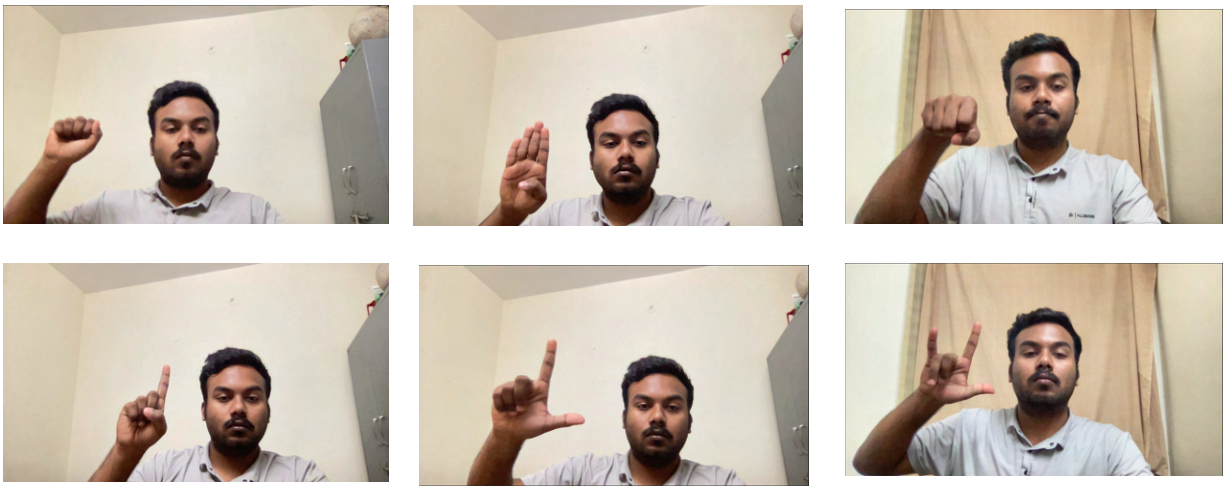
Group Members: Rahul Barodia (B20CS047)
Agnibha Burman Roy (B20CS099)

Model 1: Using SVM

We developed a Sign Language Detection model using machine learning. The project comprises four code files to collect sign language images through a webcam, preprocess the data using the MediaPipe library, train a Support Vector Machine (SVM) classifier, and implement real-time inference for sign language interpretation.

collect_imgs.py :

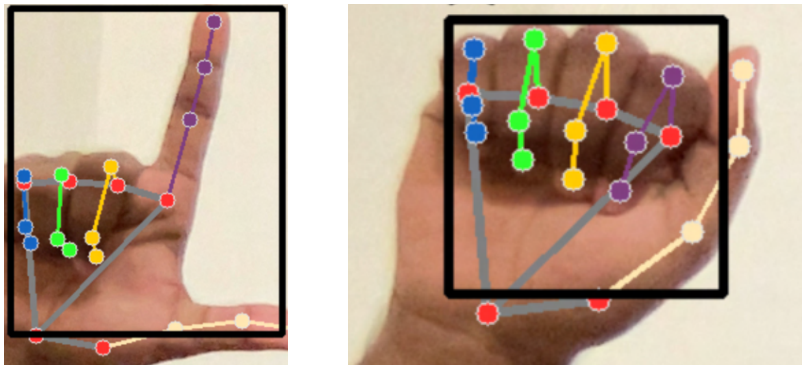
The collect_imgs.py script is designed to capture images for a sign language dataset using a webcam. It organises the collected data into folders based on different sign language classes. The script first sets up the data directory and webcam capture. It then prompts the user to prepare for data collection, displaying a message on the captured frames. The user triggers the image capture by pressing 'Q'.



A dedicated folder is created within the data directory for each sign language class. The script enters a loop, capturing and saving images until the specified dataset size is reached. During this process, real-time feedback is provided to the user through the webcam feed. The resulting dataset consists of images labelled according to their respective sign language classes, forming a foundation for training a machine learning model for sign language recognition. This script serves as the initial step in creating a comprehensive and diverse dataset for the subsequent model training process.

create_dataset.py :

We use the MediaPipe library and OpenCV to preprocess images and extract hand landmarks for a sign language detection model.
The script reads images from a specified directory, converts them to RGB format, and utilises the MediaPipe Hands module to detect hand landmarks.
For each detected hand in an image, the (x, y) coordinates of 21 keypoints are recorded.



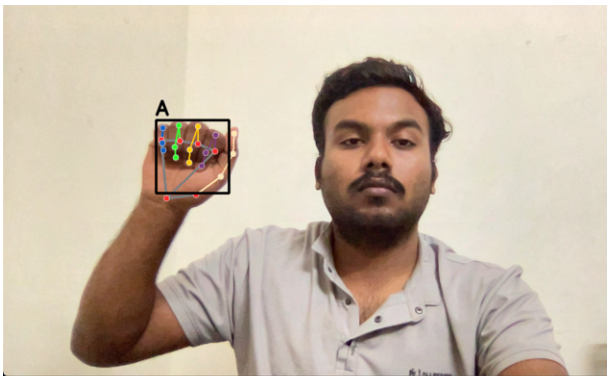
These coordinates are then flattened into a one-dimensional array and added to the dataset.
The script iterates through all the images in the dataset directory, capturing hand landmarks and corresponding labels (sign language classes).
Finally, the dataset, consisting of flattened hand landmarks and labels, is saved in a pickle file ('data.pickle') for further use in training the sign language detection model.
This preprocessing step is crucial for creating a structured and usable dataset for machine learning model training

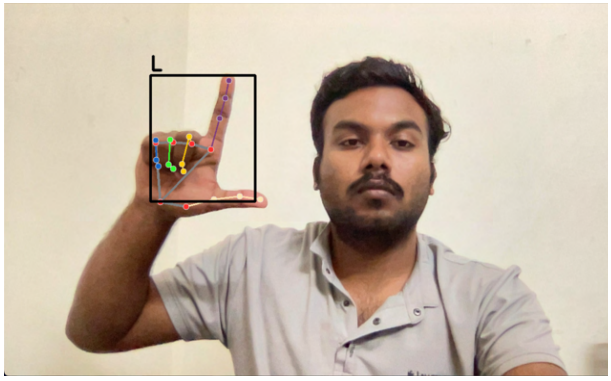
train_classifier.py :

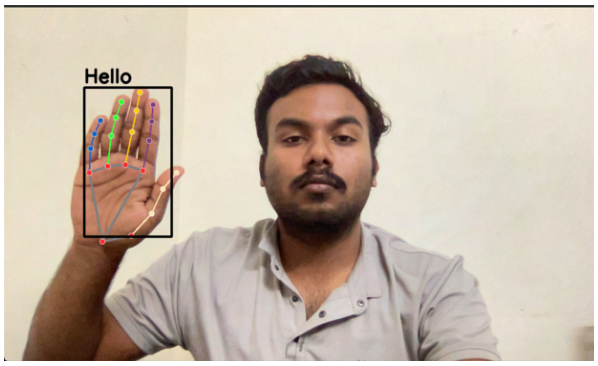
We employ a Support Vector Machine (SVM) to train a sign language detection model using the scikit-learn library.
After loading preprocessed data from a pickle file, we split it into training and testing sets. A parameter grid is defined to search for the optimal hyperparameters for the SVM model using GridSearchCV.
The model is then trained with the best parameters obtained from the grid search. Subsequently, the model's performance is evaluated on the test set, and the accuracy score is printed.
Finally, the trained SVM model with the optimal parameters is saved for future use.
This script not only automates the hyperparameter tuning process but also ensures that the model achieves the best possible accuracy on unseen data, enhancing its robustness for real-world sign language interpretation.

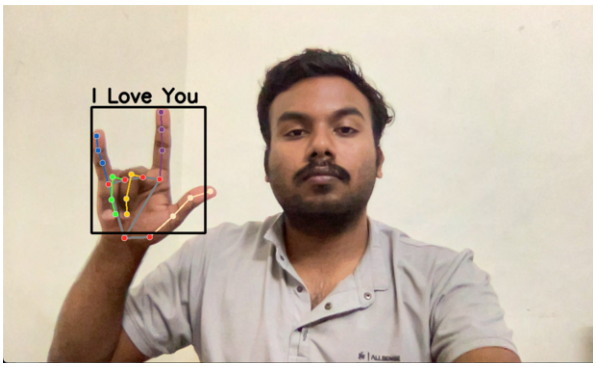
inference_classifier.py :

The "inference_classifier.py" script utilises the trained SVM model to perform real-time sign language interpretation using a webcam. The code leverages the MediaPipe library to detect and extract hand landmarks from each frame.
These landmarks are then normalised to a consistent scale. The normalized data is fed into the SVM model for prediction, and the resulting sign language gesture is displayed on the webcam feed.
The script dynamically adjusts the bounding box around the detected hand, enhancing the visual representation.
The predicted sign is overlaid on the video stream in real-time, providing an immediate and user-friendly sign language interpretation system.






Hello


I Love You

Model 2:Using LSTM

1. Introduction

The Keypoint Classifier project aims to develop a machine learning model using Long Short-Term Memory (LSTM) networks to classify keypoints based on (x, y) coordinates. The model is trained on a dataset containing sequences of keypoints, and TensorFlow Lite is employed for deployment on resource-constrained devices. This report provides a comprehensive overview of the project, including dataset preparation, model architecture, training, evaluation, and deployment.

2. Dataset

The dataset (keypoint.csv) consists of (x, y) coordinates representing keypoints. Each sequence is associated with a class label, defining the type of keypoint. The dataset is split into training and testing sets using the train_test_split function.

3. Model Architecture

3.1 LSTM Model

- Input Layer: Reshape layer to match the LSTM input shape (21 sequences with 2 coordinates each).
- LSTM Layer: 20 units with ReLU activation to capture temporal dependencies.
- Dropout Layer: 40% dropout rate to prevent overfitting.
- Dense Layers: Two dense layers with ReLU and softmax activations for classification.

3.2 Model Compilation

- Optimizer: Adam optimizer.
- Loss Function: Sparse categorical cross entropy.
- Metrics: Accuracy.

4. Model Training

- The model is trained on the training set for a maximum of 1000 epochs with a batch size of 128.

- ModelCheckpoint and EarlyStopping callbacks are used for saving the best model and preventing overfitting.

5. Model Evaluation

- The trained model is evaluated on the test set, and validation loss and accuracy are reported.

6. Model Inference and Confusion Matrix

- Inference tests are performed on both the original and TensorFlow Lite models.
- A confusion matrix is generated to analyze the model's performance, including precision, recall, and F1-score.

7. TensorFlow Lite Model Conversion

- The best model is saved and converted to TensorFlow Lite format for deployment.
- Quantization is applied to reduce the model size and improve inference speed.

8. Deployment

- The TensorFlow Lite model is ready for deployment on mobile and edge devices.
- Inference tests on the TensorFlow Lite model demonstrate its efficiency on resource-constrained platforms.

9. Results and Conclusion

- The model achieves high accuracy on the test set.
- The confusion matrix provides insights into specific misclassifications.
- The TensorFlow Lite model maintains reasonable accuracy with reduced model size.
- The project demonstrates the successful application of LSTM networks for keypoint classification, with potential applications in real-time systems.

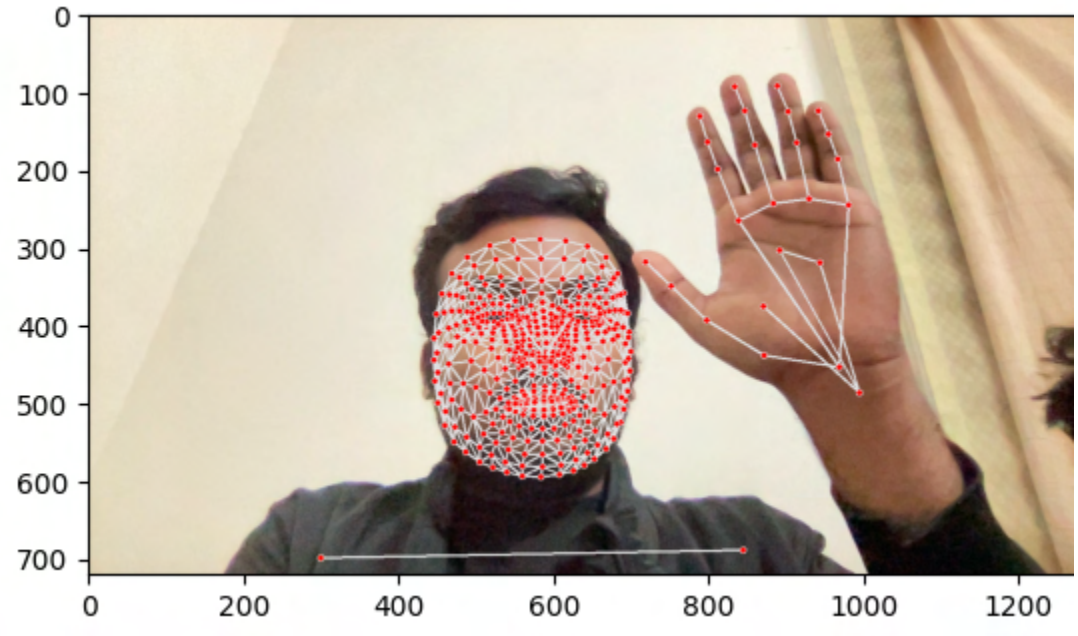
Model 3: LSTM layers to decode gestures from video sequences

1. Importing Libraries
We start by installing and importing essential libraries such as TensorFlow, OpenCV, MediaPipe, scikit-learn, and matplotlib. These libraries provide the tools we need for deep learning, computer vision, and visualization.

2. Initializing MediaPipe and OpenCV
We set up MediaPipe for holistic detection, which includes face, pose, and hand landmarks. OpenCV is used to capture video frames and display the real-time feed.

3. Real-time Sign Language Detection Loop
Within a loop that reads video frames from the camera feed, we use MediaPipe to detect holistic keypoints. The keypoints are then visualized on the frame using custom styling.

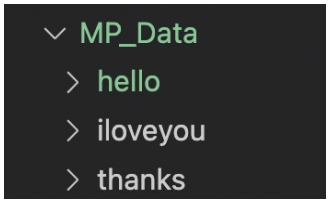
4. Landmark Extraction and Visualization
We extract landmarks from different parts of the body (pose, face, left hand, right hand) and visualise them using matplotlib.



5. Key Points Extraction
We extract key points for each body part and save them into a NumPy array. These key points serve as the input for our action detection model.

6. Dataset Preparation

To train our model, we begin by creating a dataset. We define three sign language actions: 'hello,' 'thanks,' and 'iloveyou.' Each action is represented by 30 sequences, and each sequence comprises 30 frames. The data is organized into folders for each action and sequence, with keypoints extracted using the previously implemented MediaPipe library.



7. Real-time Data Collection

Next, we extend our initial video capture loop to facilitate the collection of real-time data. The updated loop captures frames from the webcam, detects holistic keypoints using the MediaPipe model, and visually annotates the keypoints on the video feed. For each frame, we export the keypoints into NumPy arrays, creating a dataset for model training. We split our dataset into training and testing sets, preparing it for model training.

8. Model Training

LSTM Layers:

We incorporate three LSTM layers, known for their proficiency in handling sequential data. Each layer has a specific configuration:

First LSTM Layer (64 units):
return_sequences=True: Necessary for sequences-to-sequences processing.
activation='relu': Rectified Linear Unit (ReLU) activation function for introducing non-linearity.
input_shape=(30, 1662): Input shape representing 30 frames with 1662 features (keypoints).

Second LSTM Layer (128 units):
return_sequences=True: Maintains the sequence information for subsequent layers.
activation='relu': Applies the ReLU activation function.

Third LSTM Layer (64 units):
return_sequences=False: Final layer in the sequence, only returning the output.

Dense Layers:
Following the LSTM layers, we add densely connected layers to perform high-level reasoning on the extracted features:

First Dense Layer (64 units):
activation='relu': ReLU activation for introducing non-linearity.

Second Dense Layer (32 units):
activation='relu': Another layer with ReLU activation.

Output Dense Layer (Number of Actions units):
activation='softmax': Softmax activation for multiclass classification.

Model Compilation:
We compile the model by specifying the optimizer, loss function, and metrics for evaluation:
optimizer='Adam': Adaptive Moment Estimation (Adam) optimization algorithm.
loss='categorical_crossentropy': Categorical crossentropy as the loss function for multiclass classification.
metrics=['categorical_accuracy']: Accuracy as the evaluation metric.

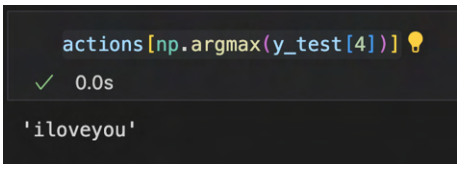
Model Training:
The model is then trained on the provided training data for 2000 epochs

Model Summary:
This summary provides a concise representation of the layers, output shapes, and trainable parameters, facilitating an understanding of the model's structure.

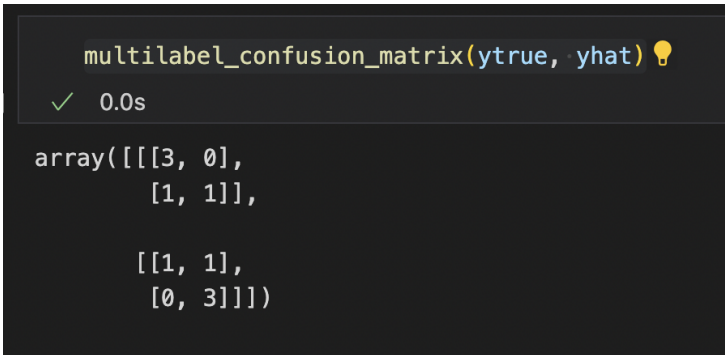
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 30, 64)            442112

 lstm_1 (LSTM)               (None, 30, 128)           98816

 lstm_2 (LSTM)               (None, 64)                49408

 dense (Dense)               (None, 64)                4160

 dense_1 (Dense)             (None, 32)                2080

 dense_2 (Dense)             (None, 3)                 99

=================================================================
Total params: 596675 (2.28 MB)
Trainable params: 596675 (2.28 MB)
Non-trainable params: 0 (0.00 Byte)
```

9. Model Evaluation

After training, we evaluate the model's performance using the test set.
We predict sign language actions using our trained model on the test set. res contains predictions, and we retrieve the predicted action for the fifth sample using np.argmax(res[4]). We also check the ground truth action for the same sample with actions[np.argmax(y_test[4])]. This comparison aids in assessing the model's accuracy in recognising sign language gestures.



We employ metrics such as multilabel confusion matrices and accuracy scores to assess how well the model generalises to unseen data.

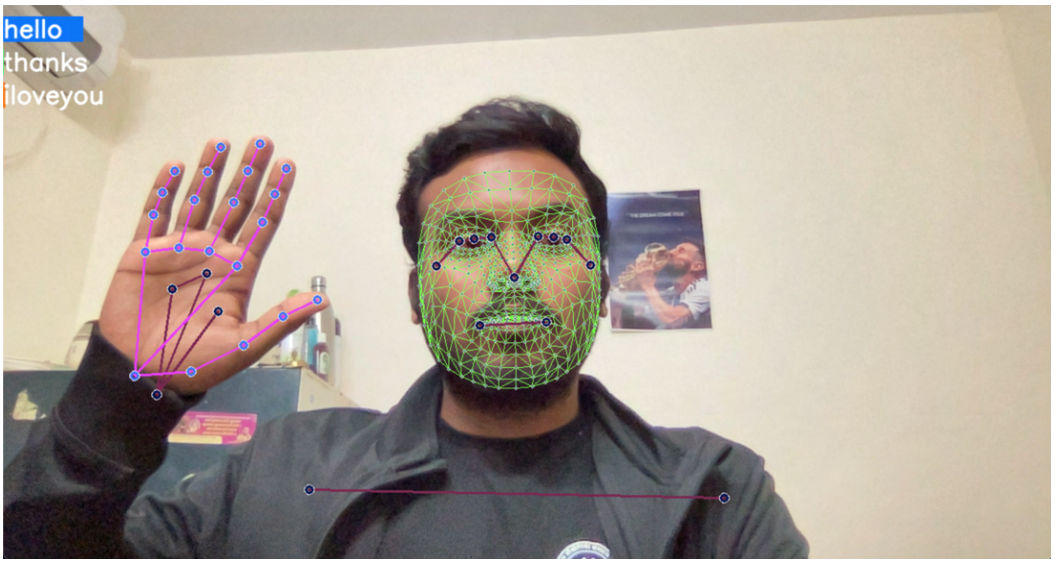


10. Real-time Sign Language Recognition

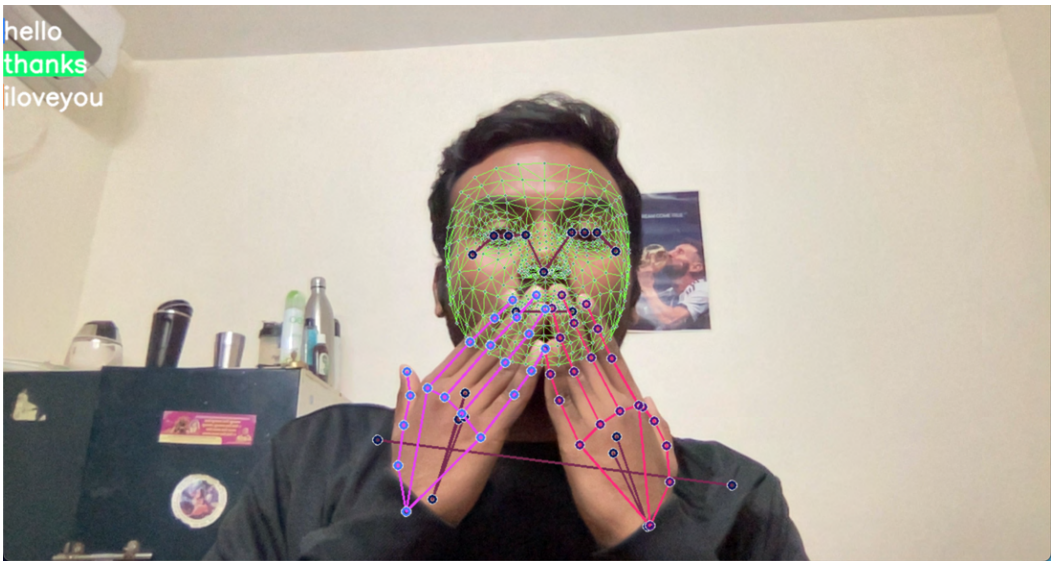
The final section introduces the real-time sign language recognition loop. Utilising the trained model, we capture video frames, detect key points, and feed sequences into the model for predictions. The predicted actions are displayed on the video feed, along with a visual representation of prediction probabilities.

Prediction Logic: Keypoints are continuously fed into the model, and predictions are made based on the last 30 frames. It is considered valid if a consistent prediction is obtained over a certain threshold.

Visualisation Logic: Predicted actions and their associated probabilities are displayed on the video feed. The last five predicted actions are shown, concisely representing the ongoing sign language message.


hello
thanks
iloveyou


hello
thanks
iloveyou


hello
thanks
iloveyou