Shatakshi Singh | 23CS30048
Srishti Somya | 23CS30055

GSSTC

GestureSpeak: Sign to Text Convertor

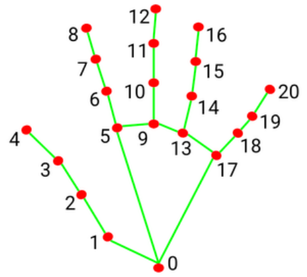
GestureSpeak

- GestureSpeak is a real-time sign language recognition system that captures hand gestures via a camera and translates them into text using computer vision and machine learning techniques. It detects and extract 3D hand landmarks, which are then processed to classify American Sign Language (ASL) gestures.
- A classifier is trained on the extracted hand landmark data to recognize specific ASL signs.
- The model and its corresponding feature scaler are saved for real-time predictions.
- The training pipeline involves dataset preprocessing, model selection, parameter tuning, and evaluation.
- The GUI processes video frames from a webcam in real time.
- Each frame undergoes hand detection and feature extraction before being passed to the trained model for classification.
- Recognized gestures are continuously appended to form words or sentences, which are displayed dynamically on the screen.
- GestureSpeak is designed to facilitate seamless communication for deaf-mute individuals, offering an efficient, user-friendly solution for everyday interactions. By combining real-time processing with an intuitive interface, it represents a step toward a more inclusive world where non-verbal communication barriers are significantly reduced.

Technologies used

1 Dataset Creation

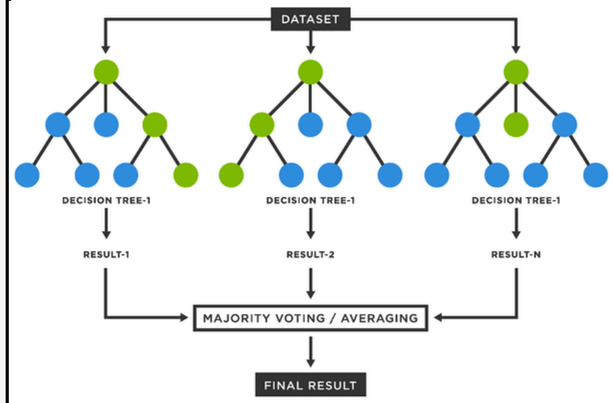
- **OpenCV** (Open Source Computer Vision Library) is used to process image data.
- **Mediapipe** is a powerful framework by Google used for real-time hand tracking and landmark detection. It provides Hand Landmarks – 21 3D coordinates representing key points on a hand.



- **Pickle** : After preprocessing, the extracted landmark data is saved using Pickle, a Python library for object serialization.

2 Training the Model

- **Pickle** is used to load the preprocessed dataset stored as .pickle files. It allows quick and easy deserialization of the data into NumPy arrays, ready for model training.
- **Scikit-Learn** : **Random Forest Classifier** has been used for GSSTC. It provides robust tools for splitting the dataset, training models, and performing cross-validation.



3 User Interface

- **Streamlit** is a Python library used to create interactive and user-friendly web applications. It provides a clean UI to visualize webcam input, display predictions, and track results in real-time.
- **Pickle** is used to load the trained model for real-time predictions. Efficient for quickly loading large machine learning models without retraining.
- **OpenCV** captures video frames from the webcam using cv2.VideoCapture(). It also allows real-time visualization using cv2.imshow().
- **Mediapipe** Hands is used for real-time hand detection and landmark extraction. It provides 3D landmarks for each hand, which are the input features for the trained model.

4 Testing and CI/CD

- **Pytest** ensures reliable testing of functions, model predictions, and system components with detailed error reporting.
- **Unittest.mock** simulates external dependencies like webcam input for controlled testing.
- **GitHub Actions** automates testing and deployment, ensuring code quality with every update.
- **Pickle** loads the trained model for prediction testing without retraining.
- **Sys** handles system-level errors for robust application testing. It provides access to system-specific parameters and functions. It is used to manage system-level tasks

Challenges faced, integration of CI/CD & tests written

- Dataset Selection: Initially, we created our own dataset, but later switched to a Kaggle dataset as it provided better accuracy.
- Model Selection: We compared XGBoost, Gradient Boost Classifier , and Random Forest using hand landmarks. Random Forest was chosen because it achieved better accuracy with less data and was more feasible.
- Handling Similar Signs: Some signs were too similar to distinguish accurately, so we introduced sub-models to improve differentiation.
- Feature Engineering: Basic hand landmarks were insufficient. To enhance accuracy, we incorporated additional features such as the angles between fingers.
- Normalization for Distance: The detected sign was affected by the distance from the camera, so we refined the normalization method to mitigate this issue.
- Initially, our model was overfitted, but we tuned the parameters to mitigate this issue.

- Automated Testing: Implemented unit tests for key components, including dataset preprocessing, model training, and the Streamlit app, to ensure system reliability.
- Version Control & Code Quality: Integrated GitHub Actions to automate linting, formatting checks, and unit tests on every commit and pull request.
- Deployment Automation: Configured CI/CD to automatically test the deployment of the Streamlit app upon successful testing, reducing manual intervention.

Thank you!