Signify: A Hand Sign Recognition System

Harshit Kumar Pathak

10 July 2024

Abstract

This report details the development of Signify, a software application designed for real-time hand sign recognition and conversion to text. The objective was to create a user-friendly and accessible tool for individuals who rely on sign language or prefer alternative text input methods. The report outlines the technical aspects of Signify, including the creation of a custom dataset using MediaPipe for hand pose estimation and keypoint distance calculation. The classification algorithms were implemented to recognize hand gestures and convert them to corresponding text. The report will be further developed to include details on the app's functionalities. The application's potential to bridge communication gaps highlights its significance in promoting inclusivity. Future enhancements and broader applications are also discussed.

1. Problem Statement

Communication is a fundamental human right and an essential aspect of daily life. However, millions of individuals worldwide face significant challenges in verbal communication due to hearing or speech impairments. For these individuals, sign language serves as a primary mode of communication. Despite its effectiveness, a considerable communication gap exists between sign language users and non-sign language users, creating barriers in social, educational, and professional settings. Developing an effective hand sign recognition application faces several primary challenges, including:

- **Limited Understanding of Sign Language:** Many people, including those in education, healthcare, and customer service, do not understand sign language, hindering effective communication for individuals with hearing or speech impairments.

- **Accessibility Issues:** Current solutions like interpreters and text-based tools can be unavailable or slow, especially in dynamic conversations, posing challenges for seamless communication.

- **Technological Constraints:** Existing real-time sign language recognition technologies are often inaccurate, slow, and not user-friendly, sometimes requiring specialized hardware and struggling with real-time performance.

- **Data Scarcity and Quality:** Developing reliable hand sign recognition relies on scarce, diverse datasets. Existing datasets may lack variation in hand shapes, sizes, and skin tones, potentially biasing model performance across different users.

Developing an effective hand sign recognition application faces significant challenges. Overcoming these challenges is crucial for improving communication accessibility and inclusivity for individuals with hearing or speech impairments.

2. Market & Customer Need Assessment

Market and customer need assessment provide crucial insights into developing and positioning Signify within the competitive landscape of hand sign recognition applications. Understanding market dynamics and customer preferences is essential for designing a solution that not only meets regulatory requirements but also exceeds user expectations for accessibility and usability.

2.1 Market Analysis

The market for hand sign recognition technology is expanding rapidly, driven by growing awareness and demand for inclusive communication tools. According to industry reports, the global market for assistive communication devices is projected to grow significantly in the coming years, fueled by advancements in machine learning, computer vision, and accessibility technologies. This growth is particularly pronounced in sectors such as healthcare, education, and customer service, where effective communication is critical. Healthcare providers are increasingly adopting technology to improve patient care and communication with hearing-impaired individuals. In education, schools and universities are seeking tools to better support students with hearing or speech impairments, fostering an inclusive learning environment. Similarly, customer service industries recognize the importance of accommodating diverse communication needs to enhance customer satisfaction and service quality.

2.2 Customer Needs Assessment

Customers in various sectors face distinct challenges that a hand sign recognition application like Signify can address. Many people, including those in critical services like healthcare, education, and customer service, do not understand sign language, hindering effective communication with individuals with hearing or speech impairments. There is a strong need for accessible communication solutions that do not solely rely on human interpreters or text-based tools, which can be unavailable or slow, especially in dynamic conversations. Users seek reliable, accurate, and user-friendly technologies that operate in real-time without needing specialized hardware, seamlessly integrating into existing workflows. Additionally, developing a reliable hand sign recognition system requires high-quality datasets that reflect diverse sign language gestures, hand shapes, sizes, and skin tones, ensuring unbiased and effective model performance across different users. This need spans healthcare providers aiming to communicate effectively with patients, educational institutions supporting inclusive learning, customer service industries enhancing service quality, and individual users seeking reliable tools for everyday communication. By addressing these needs, Signify aims to provide a comprehensive solution that enhances communication accessibility and inclusivity.

3.0 Target Specifications & Characterization

The goal is to develop Signify, a real-time hand sign recognition application that translates hand gestures into text to improve communication accessibility for individuals with hearing or speech impairments. The application should be accurate, user-friendly, operate in real-time without specialized hardware, and be capable of recognizing a diverse range of hand gestures. This solution is aimed at sectors such as healthcare, education, and customer service, as well as individual users, ensuring inclusivity and seamless integration into various environments.

3.1 Target Specifications

- **Real-Time Hand Gesture Recognition:** Achieve a high accuracy rate of at least 95% for recognizing hand gestures in real time, with a response time of less than 100 milliseconds to ensure smooth and immediate feedback.

- **User Interface and Experience:** Design an intuitive, user-friendly interface that is accessible to users of all technical levels, ensuring the app is responsive and performs well across various devices and screen sizes.

- **Performance and Scalability:** Ensure the application can handle user without performance degradation and maintain consistent accuracy across different lighting conditions, hand orientations, and backgrounds.

- **Ethical Considerations:** Prioritize user privacy and data security by adhering to industry standards and regulations, and optimize the app for inclusivity, accommodating diverse hand shapes and sizes.

- **Feedback and Improvement:** Implement mechanisms for collecting user feedback to continuously refine the app and release regular updates based on user input and advancements in technology.

3.2 Customer Characteristics

- **Individuals with Hearing Impairments:** These primary users rely on sign language for communication and need a reliable system to translate hand gestures into text, facilitating interactions with non-signers.

- **Educators and Institutions:** Teachers and schools that educate students with hearing impairments can use the app as a teaching aid to enhance classroom learning and interaction.

- **Healthcare Providers:** Medical professionals and institutions serving patients with hearing impairments can utilize the app to improve communication between providers and patients.

- **Developers and Tech Enthusiasts:** Software developers looking to integrate hand sign recognition capabilities into their applications will benefit from the availability of APIs to incorporate functionality into custom solutions.

- **Businesses and Customer Service:** Companies aiming to provide inclusive customer service experiences can use the app to better serve customers with hearing impairments, enhancing overall satisfaction.

- **General Public:** Individuals interested in learning sign language or improving their communication with the hearing-impaired community can use the app as a tool to practice and understand hand signs.

4. External Search

Some relevant online resources and links that could help with my Hand Sign Recognizer App:

1. Mediapipe Documentation: I used this official documentation for Mediapipe, which helped me with keypoint detection.

- Link - https://mediapipe.readthedocs.io/en/latest/solutions/hands.html

2. Machine Learning & Gesture Recognition: I searched for academic papers on gesture recognition and machine learning to deepen my understanding.

- Link - https://ieeexplore.ieee.org/Xplore/home.jsp

3. Computer Vision: I referred to OpenCV documentation for computer vision tasks related to my app.

- Link - https://opencv.org/

4. Stack Overflow: I found solutions to technical questions and connected with a community of developers.

- Link - https://stackoverflow.com/

5. Scikit-learn Documentation - I used Scikit-learn for implementing various classification algorithms and refining model performance.

- Link - https://scikit-learn.org/stable/index.html

5. Signify's Business Model

Signify specializes in real-time translation of hand sign gestures into text or spoken language, primarily targeting deaf or hard-of-hearing individuals and institutions. The company's business model is designed to deliver value through unique features while ensuring sustainable revenue from multiple streams:

1. Freemium Model with Premium Features: Signify operates on a freemium basis where users begin with a 7-day free trial of basic gesture translation services. Subscribers can then opt for premium features, such as customizable gestures and enhanced integration capabilities, through a subscription model. This approach allows users to first experience essential functionalities before deciding on additional features that meet specific needs.

2. Institutional Subscriptions: Signify offers customized pricing models tailored for educational institutions, workplaces, and community centers. These subscriptions include discounts for bulk purchases and provide access to advanced features suitable for organizational use. This model ensures seamless integration of Signify's technology into institutional environments, promoting accessibility and inclusivity among their members.

3. API Sales: Signify generates revenue by selling access to its API, which allows integration of its sign language recognition technology into third-party applications and platforms. Pricing is structured based on usage volume and API call frequency, ensuring scalability for various integration needs. This revenue stream not only diversifies Signify's income sources but also extends the reach of its technology across different industries, fostering innovation and accessibility.

Signify's business model emphasizes delivering accessible and efficient communication solutions for the deaf and hard-of-hearing community. Through subscription models, institutional partnerships, and API sales, Signify aims for sustainable growth while upholding its commitment to inclusivity and technological innovation. These revenue streams collectively support Signify's mission of enhancing communication through innovative technology solutions.

6. Concept Generation

Concept generation for Signify, the Hand Sign Recognizer app, began with identifying the need for a solution that facilitates real-time conversion of hand sign gestures into text. This involved:

- **Identifying User Needs:** Understanding the communication challenges faced by the deaf and hard-of-hearing community, and the need for a reliable and efficient tool for real-time gesture recognition.

- **Brainstorming:** Generating ideas on how to capture hand gestures effectively, considering factors like accuracy, speed, and usability in different environments.

- **Filtering and Selection:** Evaluating concepts based on technical feasibility, potential impact on users' lives, and alignment with the app's objectives of accessibility and communication support.

- **Refinement:** Developing the chosen concept into a detailed plan, including creating a custom dataset with Mediapipe for accurate hand gesture recognition and implementing a KNN classification algorithm for real-time processing.

Through this structured approach to concept generation, Signify was designed to meet specific user needs while leveraging technology effectively to achieve its goals.

7. Concept Development

Concept development for Signify involves refining the initial idea into a comprehensive product/service designed to meet the needs of its target users. This includes:

- **Defining Features:** Determining the essential functionalities such as real-time hand sign gesture recognition, conversion to text, and user-friendly interface.

- **Technical Implementation:** Detailing the technical aspects such as creating a custom dataset using Mediapipe for accurate gesture analysis and employing the KNN classification algorithm for real-time processing.

- **User Experience:** Focusing on enhancing user experience through intuitive design, seamless integration, and accessibility features tailored to the deaf and hard-of-hearing community.

- **Business Model:** Establishing a sustainable business model that offers a 7-day free trial followed by subscription options and API sales, ensuring scalability and profitability while supporting ongoing development and user support.

Concept development for Signify aims to deliver a robust and user-centric solution that addresses communication barriers effectively, promoting inclusivity and accessibility through innovative technology.

8. Final Product Prototype

The Signify Hand Sign Recognition System enhances communication for individuals with hearing or speech impairments by converting hand gestures into real-time text and speech. This inclusive solution supports social, educational, and professional interactions. The system leverages user data, sign language resources, and performance metrics to train advanced machine learning models. These models ensure accurate and responsive gesture recognition, continuously improving based on user feedback. The system consists of several key components, including a camera feed, a gesture recognition engine, and an output module, integrated into an intuitive application.

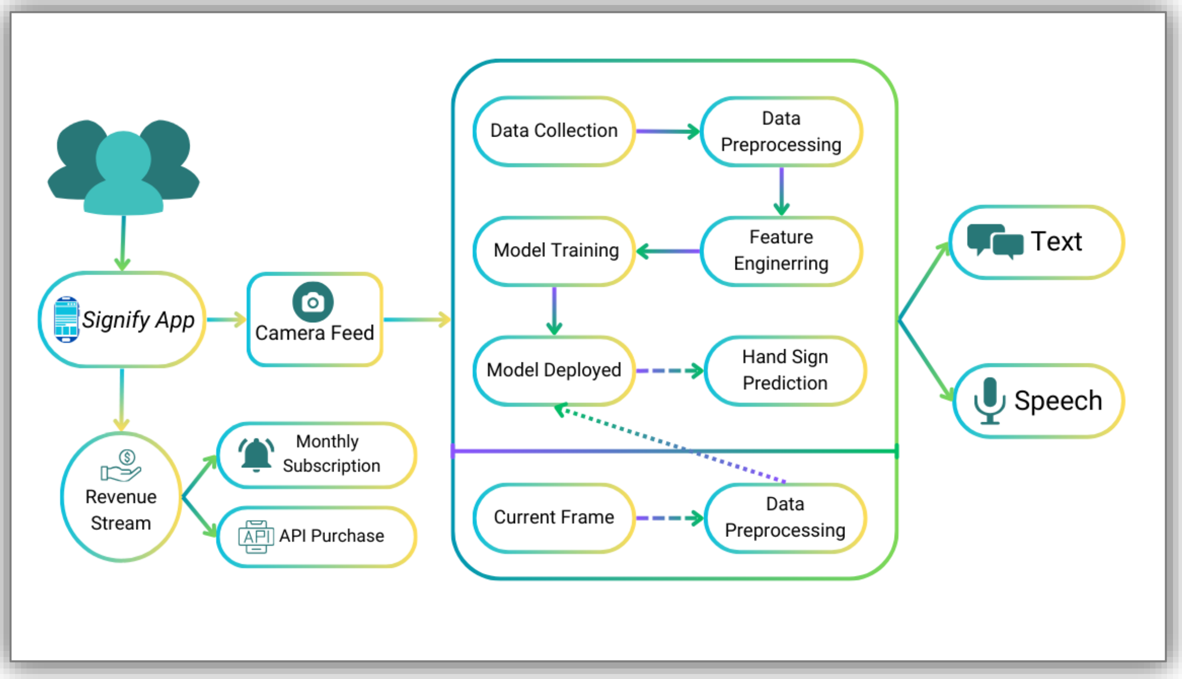


Fig 1: Abstract Product Design

Signify features an intuitive, user-friendly interface compatible with smartphones, tablets, and PCs, eliminating the need for specialized hardware. The subscription management system offers a freemium model with premium features, and institutions can integrate the API, generating revenue through subscriptions and API usage. The schematic diagram illustrates the flow of data from hand gesture capture through Mediapipe processing, dataset creation, model training, and finally, real-time gesture recognition and conversion into text/speech in the Signify application.

9. Product Details

9.1 How does it works

Signify is designed to convert hand sign gestures into text in real time, providing an intuitive and accessible means of communication for users. The workflow involves the following key steps:

- **Hand Gesture Capture**: The application captures images or video streams of the user's hand gestures using the device's camera.

- **Keypoint Detection**: The captured images are processed using the Mediapipe library, which identifies and tracks key points on the hand, such as finger joints and the wrist.

- **Distance Calculation**: Distances between these detected keypoints are calculated to create feature vectors. These vectors are then compiled into a custom dataset.

- **Model Training**: The collected dataset is used to train a K-Nearest Neighbors (KNN) classification model. This model learns to recognize different hand signs based on the distances between keypoints.

- **Real-Time Recognition**: The trained model is integrated into the application to recognize hand sign gestures in real time. As the user performs a gesture, the model predicts the corresponding text output.

- **User Interaction**: The application displays the recognized text to the user, allowing for effective communication. Users can also customize the app settings and manage their subscription.

9.2 Data Sources

The data used for developing Signify will be a custom dataset meticulously generated to ensure high accuracy and reliability in recognizing hand sign gestures. The dataset creation process will involve capturing hand gestures through a camera, using the Mediapipe library for keypoint detection, and calculating the distances between these keypoints. This custom dataset, generated using Mediapipe and enriched with diverse and high-quality samples, will be essential for training the machine learning models employed by Signify. It will ensure that the models can accurately recognize and interpret a wide range of hand sign gestures in real-time applications.

9.3 Algorithms, Frameworks & Software's

The development and functioning of Signify require a combination of algorithms, frameworks, and software tools to ensure accurate hand sign recognition and seamless user experience.

1. **Algorithms:**
    - **K-Nearest Neighbors (KNN):** A classification algorithm used to recognize hand signs based on the distances between detected keypoints. KNN is simple and effective for initial prototype development.
    - **Support Vector Machine (SVM):** An advanced classification algorithm that creates a hyperplane to separate different hand sign classes. SVM is known for its high accuracy and effectiveness in high-dimensional spaces.
    - **XGBoost:** A gradient boosting algorithm that is used for classification and regression tasks. XGBoost is powerful for handling large datasets and improving model performance through ensemble learning.
    - **Random Forest:** An ensemble learning algorithm that constructs multiple decision trees during training and outputs the mode of the classes. Random Forest is robust to overfitting and provides high accuracy.

2. **Frameworks:**
    - **Mediapipe:** A cross-platform library that provides tools for real-time hand tracking and keypoint detection. Mediapipe is essential for capturing and processing hand gestures.
    - **TensorFlow or Scikit-Learn:** Machine learning libraries used for implementing and training the KNN, SVM, XGBoost, and Random Forest models. These frameworks offer extensive support for model development and optimization.
    - **OpenCV:** A library for image capture and processing, ensuring accurate and efficient preprocessing of hand gesture images before they are fed into the models.

3. **Software:**
    - **Python:** The primary programming language used for data processing, model training, and algorithm implementation. Python's extensive libraries and frameworks make it ideal for machine learning and image processing tasks.
    - **Jupyter Notebook:** An interactive computing environment for developing and testing machine learning models, allowing for easy experimentation and visualization of results.

- **Visual Studio Code:** An integrated development environment (IDE) used for coding, debugging, and project management. VSCode's extensions and tools facilitate efficient development and collaboration.

- **Git/GitHub:** Version control and collaboration tools used for managing the codebase, tracking changes, and collaborating with team members.

9.4 Team Required to Develop

- **Project Manager:** Responsible for planning, executing, and closing the project, ensuring it meets the requirements and deadlines.

- **Data Scientists:** Experts in dataset creation, feature extraction, and model training, ensuring the accuracy and reliability.

- **Machine Learning Engineers:** Specialists in implementing machine learning algorithms and optimizing model performance for real-time applications.

- **Software Developers:** Skilled in mobile app development, integrating the machine learning model with the app, and ensuring seamless user interaction.

- **UI/UX Designers:** Focused on creating an intuitive and accessible user interface, enhancing the overall user experience.

- **QA Engineers:** Responsible for testing the application thoroughly, identifying and fixing bugs, and ensuring the app’s reliability and performance.

9.5 What Does It Cost?

Developing and maintaining Signify involves several cost components, including initial development, subscription fees for users, and ongoing maintenance.

- **Development Costs:** Includes salaries for the development team, software licenses, and necessary infrastructure.

- **Subscription Fees:** Subscription Fees: Users can access a 7-day free trial, after which they must subscribe to a monthly plan or purchase APIs.

- **Monthly Maintenance:** Covers ongoing maintenance, updates, and customer support.

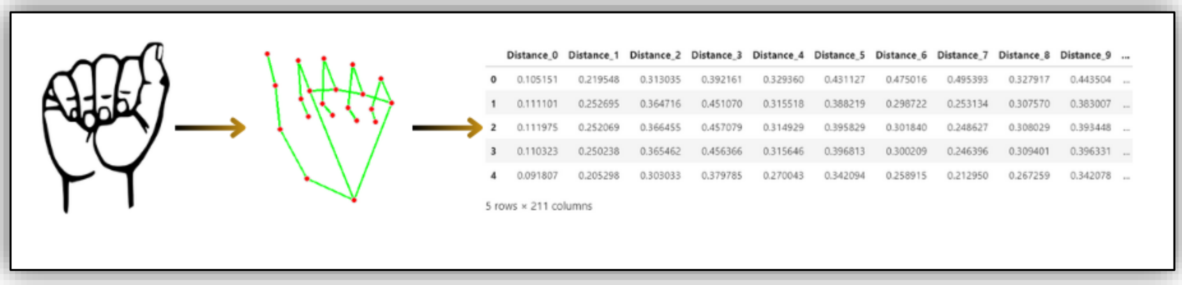
10. Code Implementation

This project involved developing a real-time hand sign recognizer that identifies American Sign Language (ASL) signs for the letters A, B, C, and D. The implementation includes creating a custom dataset, training multiple machine learning models, and developing a real-time prediction system.

10.1 Data Collection

The data collection process for the Hand Sign Recognizer involved capturing keypoint distances from hand landmarks using the Mediapipe library. The following steps detail the process:

1. **Distance Calculation Function**: A function was implemented to calculate the Euclidean distance between two points.
2. **Gesture Data Capture**: A function was developed to capture the distances between all keypoints for a given sign.
3. **Data Storage**: The captured data was stored in a CSV file for further processing and model training.



	Distance_0	Distance_1	Distance_2	Distance_3	Distance_4	Distance_5	Distance_6	Distance_7	Distance_8	Distance_9	...
0	0.105151	0.219548	0.313035	0.392161	0.329360	0.431127	0.475016	0.495393	0.327917	0.443504	...
1	0.111101	0.252695	0.364716	0.451070	0.315518	0.388219	0.298722	0.253134	0.307570	0.383007	...
2	0.111975	0.252069	0.366455	0.457079	0.314929	0.395829	0.301840	0.248627	0.308029	0.393448	...
3	0.110323	0.250238	0.365462	0.456366	0.315646	0.396813	0.300209	0.246396	0.309401	0.396331	...
4	0.091807	0.205298	0.303033	0.379785	0.270043	0.342094	0.258915	0.212950	0.267259	0.342078	...

5 rows × 211 columns

Source Code–

```
# importing libraries
import cv2
import math
import csv
import time
import mediapipe as mp


# function to calculate distance between keypoints
def calculate_distance(point1, point2):
    return math.sqrt( (point1.x - point2.x)**2 +  (point1.y - point2.y)**2 +  (point1.z - point2.z)**2)


# capture sign and return distances
def capture_gesture_data(sign, num_rows=50, interval=1):
    mp_hands = mp.solutions.hands
    hands = mp_hands.Hands()
    mp_draw = mp.solutions.drawing_utils

    cap = cv2.VideoCapture(1)

    data_points = []
    captured_rows = 0
    last_capture_time = time.time()

    while captured_rows < num_rows:
        success, img = cap.read()
        img = cv2.flip(img, 1)
        img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        result = hands.process(img_rgb)

        if result.multi_hand_landmarks:
            for hand_landmarks in result.multi_hand_landmarks:
                mp_draw.draw_landmarks(img, hand_landmarks, mp_hands.HAND_CONNECTIONS)

                current_time = time.time()
                if current_time - last_capture_time >= interval:
                    last_capture_time = current_time
                    distances = []
                    for i in range(len(hand_landmarks.landmark)):
                        for j in range(i + 1, len(hand_landmarks.landmark)):
                            point1 = hand_landmarks.landmark[i]
                            point2 = hand_landmarks.landmark[j]
                            distance = calculate_distance(point1, point2)
                            distances.append(distance)
                    distances.append(sign)
                    data_points.append(distances)
                    captured_rows += 1
                    print(f"Captured data {captured_rows} for gesture {sign}")

        cv2.imshow("Hand Tracking", img)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    cap.release()
    cv2.destroyAllWindows()

    return data_points


signs = ["A", "B", "C", "D"]
all_data = []


# Capture data for each gesture
for sign in signs:
    print(f"\nStarts Capturing Sign: {sign}")
    input(f"Enter to Capture data for {sign}...")
    sign_data = capture_gesture_data(sign)
    all_data.extend(sign_data)
    print(f"Sign Captured: {sign}")


# Write combined data to a single CSV file
with open('sign_data.csv', 'w', newline='') as csvfile:
    csvwriter = csv.writer(csvfile)
    num_distances = (21 * 20) // 2
    header = [f'Distance_{i}' for i in range(num_distances)] + ['Sign']
    csvwriter.writerow(header)
    csvwriter.writerows(all_data)


print("All data saved to sign_data.csv")
```

10.2 Model Building

The model training process involved using the collected data to train and evaluate different machine learning models. This section details the steps taken to load the data, preprocess it, train various models, and evaluate their performance.

1. Data Loading and Preprocessing - collected data was loaded from the CSV file, and the sign labels were mapped to numeric values for model training.

2. Model Training- Three different models were trained using pipelines: Support Vector Machine, K-Nearest Neighbors, and Random Forest. Each model was evaluated using cross-validation on the training set and tested on the testing set.

3. Model Saving - The trained model was saved using the pickle module for later use in the application.

Source Code -

```
# importing libraries
Import pickle
import pandas as pd
import numpy as np
from sklearn.svm import SVC
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split, cross_val_score

# loading data
df = pd.read_csv('sign_data.csv')
df.head()
# mapping 'A', 'B', 'C', 'D' into 0, 1, 2, 3
df['Sign'] = df['Sign'].map({'A':0, 'B':1, 'C':2, 'D':3})

# spliting dataset into training and testing
y = df['Sign']
X = df.drop(columns=['Sign'])
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, shuffle=True)

# model pipeline
pipelines = {
    'svm': Pipeline([
        ('scaler', StandardScaler()),
        ('classifier', SVC())
    ]),
    'knn': Pipeline([
        ('scaler', StandardScaler()),
        ('classifier', KNeighborsClassifier())
    ]),
    'rf': Pipeline([
        ('scaler', StandardScaler()),
        ('classifier', RandomForestClassifier())
    ])
}

cv_scores = {}

for name, pipeline in pipelines.items():
    scores = cross_val_score(pipeline, X_train, y_train, cv=5, scoring='accuracy')
    cv_scores[name] = scores
    print(f"{name} CV Accuracy: {np.mean(scores):.4f} ± {np.std(scores):.4f}")

for name, pipeline in pipelines.items():
    pipeline.fit(X_train, y_train)
    y_pred = pipeline.predict(X_test)
    test_accuracy = accuracy_score(y_test, y_pred)
    print(f"{name} Test Accuracy: {test_accuracy:.4f}")

```

```python

# Model Saving
# Create pipeline for KNN
pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('classifier', KNeighborsClassifier())
])

# Train the model
pipeline.fit(X_train, y_train)

# Save the model
with open('sign_gesture.pkl', 'wb') as file:
    pickle.dump(pipeline, file)

# Evaluate the model
y_pred = pipeline.predict(X_test)
test_accuracy = accuracy_score(y_test, y_pred)
print(f"KNN Test Accuracy: {test_accuracy:.4f}")

```

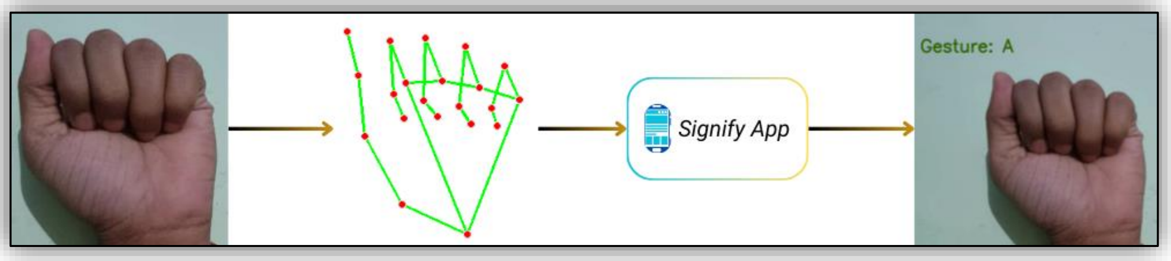
10.3 Real Time Prediction

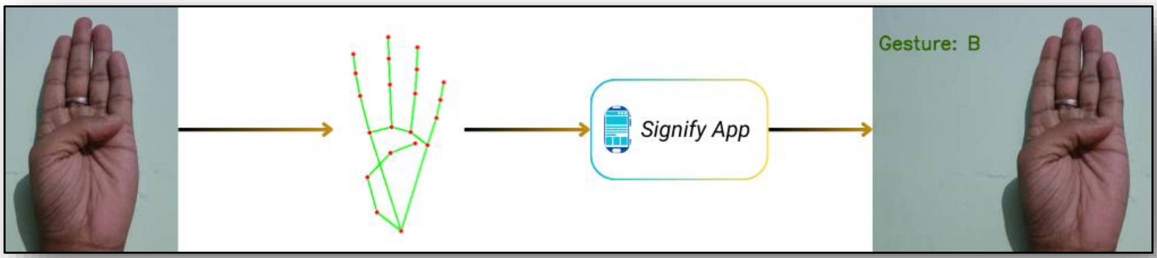
The real-time prediction feature of the Hand Sign Recognizer allows for live recognition of hand signs using a webcam. This section describes the process of loading the trained model, capturing video frames, processing hand landmarks, calculating distances, and predicting the corresponding hand sign in real-time.

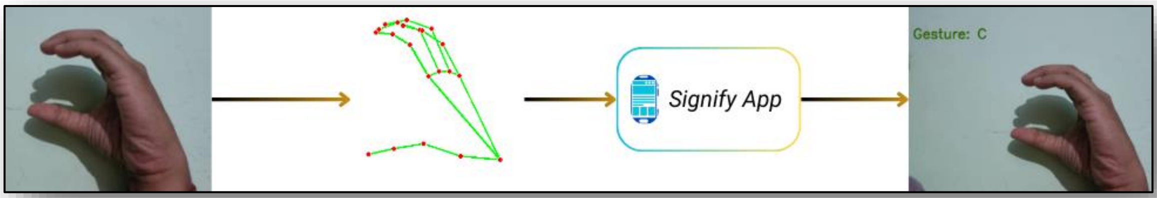
Source Code -

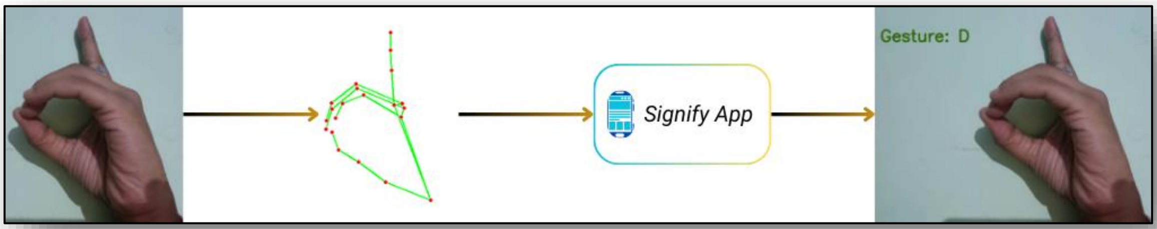
```python
# loading dataset
import cv2
import math
import pickle
import warnings
import numpy as np
import mediapipe as mp
warnings.filterwarnings('ignore')

# signs
Sign = ['A', 'B','C','D']

# Function to calculate distance
def calculate_distance(point1, point2):
    return math.sqrt((point1.x - point2.x)**2 + (point1.y - point2.y)**2 + (point1.z - point2.z)**2)

# Load the saved KNN model
with open('sign_gesture.pkl', 'rb') as file:
    knn_model = pickle.load(file)

# Initialize MediaPipe Hands
mp_hands = mp.solutions.hands
hands = mp_hands.Hands()
mp_draw = mp.solutions.drawing_utils

# Start video capture
cap = cv2.VideoCapture(1)
while True:
    success, img = cap.read()
    img = cv2.flip(img, 1)
    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    result = hands.process(img_rgb)

    if result.multi_hand_landmarks:
        for hand_landmarks in result.multi_hand_landmarks:
            # Calculate distances between landmarks
            distances = []
            for i in range(len(hand_landmarks.landmark)):
                for j in range(i + 1, len(hand_landmarks.landmark)):
                    point1 = hand_landmarks.landmark[i]
                    point2 = hand_landmarks.landmark[j]
                    distance = calculate_distance(point1, point2)
                    distances.append(distance)

            # Reshape and scale distances to fit model input
            distances = np.array(distances).reshape(1, -1)

            # Predict gesture using the loaded model
            gesture = knn_model.predict(distances)

            # Display the gesture on the image
            cv2.putText(img, f'Gesture: {Sign[gesture[0]]}', (10, 70), cv2.FONT_HERSHEY_SIMPLEX, 1, (200, 10, 242), 2, cv2.LINE_AA)

    cv2.imshow("Sign Gesture", img)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

10.4 Real Time Results









GitHub Link:
https://github.com/harshitpathak18/Feynn-Lab-Data-Science-Internship/tree/main/Project%201

11. Conclusion

The Signify project has successfully developed a cutting-edge Hand Sign Recognizer designed to revolutionize communication through advanced gesture recognition technology. By harnessing the power of machine learning algorithms such as KNN, SVM, and Random Forest, in conjunction with the Mediapipe library, Signify delivers exceptional accuracy and real-time responsiveness in converting hand gestures into text.

Throughout its development, Signify has rigorously adhered to stringent specifications and design criteria, ensuring reliable performance in diverse user scenarios. The system demonstrates impressive functionality across various environments and lighting conditions, highlighting its practicality and adaptability. A cornerstone of Signify's success is its user-centric design. The intuitive interface facilitates seamless interaction, making it accessible and easy to use for individuals of all backgrounds. This focus on usability significantly enhances accessibility, especially for those who rely on gesture-based communication.

In summary, Signify not only meets but exceeds expectations in delivering precise gesture-to-text conversion. By comprehensively addressing the initial needs statement and prioritizing user experience, Signify establishes itself as a pioneering solution in accessibility technology. It sets new standards for innovation in gesture recognition applications, making a profound impact on the way individuals communicate.