



City University of London

School of Science & Technology

MSc Data Science

Project Report

2023/24

Sign Language Recognition using Machine Learning and Computer Vision

Baibhav Datta

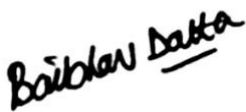
Supervised by: Professor Artur d'Avila Garcez

December 2024

Declaration

By submitting this work, I declare that this work is entirely my own except those parts duly identified and referenced in my submission. It complies with any specified word limits and the requirements and regulations detailed in the assessment instructions and any other relevant programme and module documentation. In submitting this work I acknowledge that I have read and understood the regulations and code regarding academic misconduct, including that relating to plagiarism, as specified in the Programme Handbook. I also acknowledge that this work will be subject to a variety of checks for academic misconduct.

Signed:

A handwritten signature in black ink, reading "Baibhav Datta". The signature is written in a cursive style, with the first name "Baibhav" and the last name "Datta" clearly visible. The signature is positioned above the printed name.

Baibhav Datta

ABSTRACT

This report presents the development of an efficient sign language recognition system capable of interpreting the letters of English alphabet in real-time through live webcam input. The system leverages computer vision and machine learning techniques to process hand gestures and predict corresponding alphabetic signs.

A comprehensive literature review was conducted to explore existing approaches to sign language recognition, informing the selection of suitable feature extraction methods. Various feature extraction techniques were explored and evaluated. The final model was trained using the set of extracted features that performed the best during evaluation. The system was integrated with a real-time webcam feed, using OpenCV to capture frames, preprocess them, and predict the alphabet sign with the trained model. The real-time testing program demonstrates robust performance, accurately predicting most letters, with only a few occasionally being misclassified.

Key Contributions:

- Engineered several novel features for hand gesture recognition, including joint angles, inter-finger angles, finger curls, and palm orientation, which significantly contributed to the success of the final model.
- Successfully combined the trained model with a webcam interface using OpenCV, and a user-friendly UI, enabling real-time recognition and seamless testing of alphabetic hand gestures.

This work offers a promising step toward enhancing communication accessibility for individuals with hearing and speech impairments.

Keywords: Sign Language Recognition, Feature Extraction, Real-Time Testing, Computer Vision

Table of Contents

1. INTRODUCTION AND OBJECTIVES.....	6
2. CONTEXT	9
2.1 OVERVIEW OF SIGN LANGUAGE RECOGNITION	9
2.1.1 Sensor-based Recognition	10
2.1.2 Vision-based Recognition	10
2.2 RELATED WORK	11
2.2.1 Shin et al. - American Sign Language Alphabet Recognition by Extracting Feature from Hand Pose Estimation	11
2.2.2 Kołodziej et al. - Using Deep Learning for Real-Time ASL Recognition	12
2.2.3 Singh et al. - Enhancing Sign Language Detection through Mediapipe and Convolutional Neural Networks (CNN)	13
2.2.4 Adeyanju et al. - Development of an American Sign Language Recognition System using Canny Edge and Histogram of Oriented Gradient	13
2.3 IMPLICATIONS FOR THE CURRENT PROJECT	13
3. METHODS	15
3.1 DATASET DESCRIPTION	15
3.2 DATA PREPROCESSING	16
3.3 FEATURE EXTRACTION METHODS	17
3.3.1 Histogram of Oriented Gradients (HOG)	17
3.3.2 Hand Landmark Extraction	19
3.3.3 Distance Features.....	21
3.3.4 Joint Angle Features	22
3.3.5 Inter-Finger Angles.....	24
3.3.6 Finger Curl	25
3.3.7 Palm Orientation	26
3.3 DATA SPLITTING	27
3.4 MODEL TRAINING	28
3.5 MODEL EVALUATION	31
3.6 REAL-TIME TESTING	32
3.6.1 Graphical User Interface (UI)	33
3.6.2 Real-Time Testing Workflow	33
3.6.3 Real-Time Testing Evaluation Setup.....	34
4. RESULTS	35
4.1 MODEL 1: HOG FEATURES + SVM	35
4.2 MODEL 2: FINGER ANGLES + INTER-FINGER ANGLES + SVM.....	36
4.3 MODEL 3: DISTANCE FEATURES + FINGER ANGLES + INTER-FINGER ANGLES + SVM.....	37
4.4 MODEL 4: FINGER ANGLES + INTER-FINGER ANGLES + FINGER CURLS + SVM	38
4.5 MODEL 5: FINGER ANGLES + INTER-FINGER ANGLES + FINGER CURLS + PALM ORIENTATION + SVM	39
4.6 USER INTERFACE FOR REAL-TIME TESTING	40
5. DISCUSSION.....	42
5.1 MODEL 1	42
5.1.1 Factors Contributing to Poor Real-Time Performance	42
5.1.2 Factors Influencing Better Performance for Certain Letters	43
5.2 MODEL 2	44

5.2.1 Data Loss in Feature Extraction	45
5.2.2 Factors Influencing the Better Real-Time Testing Performance Compared to Model 1	45
5.3 MODEL 3	46
5.3.1 Factors Contributing to Poor Real-Time Performance	47
5.3.2 Factors Influencing Correct Prediction of Certain Letters	47
5.4 MODEL 4	48
5.4.1 Improved Real-Time Robustness with Finger Curl	49
5.4.2 Struggling Letters in Real-Time: I, R, and U	49
5.5 MODEL 5	50
5.5.1 Palm Orientation Captures Global Hand Pose	50
5.5.2 The Gestures I and J	51
5.5.3 Enhanced Generalization to Real-Time Testing	51
5.6 CONCLUSION AND INSIGHTS FROM THE DISCUSSION	51
6. EVALUATION, REFLECTIONS AND CONCLUSIONS	53
6.1 EVALUATION OF THE PROJECT	53
6.2 GENERAL CONCLUSIONS AND CONTRIBUTION	54
6.3 IMPLICATIONS AND FURTHER WORK	55
6.4 REFLECTIVE EVALUATION	55
6.5 FINAL CONCLUSION	56
GLOSSARY	57
REFERENCES	59
APPENDIX A: PROJECT PROPOSAL	60
APPENDIX B: CODE FOR THE BEST PERFORMING MODEL	72
APPENDIX C: PROMPTS USED WITH CHATGPT	80
APPENDIX D: REAL-TIME TESTING SCREENSHOTS	81

1. Introduction and Objectives

In recent years, the use of machine learning and computer vision has enabled significant advancements in assistive technology, particularly for improving communication accessibility for individuals with hearing and speech impairments. Sign language serves as a primary mode of communication for many in the deaf and mute communities, facilitating interaction through hand gestures that convey letters, words, and phrases. However, a lack of widespread proficiency in sign language among the general population can create communication barriers. A robust, real-time sign language recognition system could bridge this gap, enabling smoother, more accessible communication for hearing-impaired individuals.

This project focuses on experimenting with different feature engineering approaches, building on a literature review of several related studies in the field of sign language recognition. The goal is to develop an efficient real-time sign language recognition system capable of interpreting alphabetic hand gestures using machine learning and computer vision techniques. Leveraging a webcam interface and a trained predictive model, the system seeks to identify hand gestures representing letters in the alphabet and display the corresponding letter in real-time. The approach explores different methods to maximize both accuracy and responsiveness, aiming to create a system suitable for dynamic, real-world applications where seamless recognition is essential.

The objectives of this project are as follows:

1. Literature Review: Conduct a comprehensive review of current sign language recognition systems to understand the strengths and limitations of various approaches, particularly in gesture recognition using computer vision.

2. Data Loading and Preprocessing: Load and preprocess the sign language dataset to prepare it for feature extraction and model training.
3. Feature Extraction: Evaluate different feature extraction methods to determine the most effective combination of features for recognizing sign language gestures.
4. Model Training and Optimization: Develop and train a machine learning model using the pre-processed dataset of sign language alphabet images.
5. Model Analysis: Perform an in-depth analysis of model performance, comparing results across different approaches and models to identify the most effective solution for real-time sign language recognition.
6. System Integration: Integrate the trained model into a real-time system that uses a webcam and OpenCV to capture live video. This will allow the model to process and recognize hand gestures in real-time, testing its performance with actual data.
7. Evaluation and Testing: Conduct extensive testing of the system's performance in real-time, analysing its accuracy, responsiveness, and robustness under varying lighting conditions and backgrounds.
8. Reflection and Conclusion: Assess the project outcomes, identify areas for improvement, and discuss the potential of such a system to improve communication for the hearing-impaired community.

By achieving these objectives, this project intends to contribute to the field of accessible technology, demonstrating a viable solution for real-time sign language recognition that could facilitate broader communication and understanding across diverse communities.

To enhance the clarity and quality of this report, I have utilized ChatGPT as a tool to assist in structuring and refining the presentation of my report. The

recommendations obtained from ChatGPT were thoughtfully integrated, contributing to improvements in phrasing, conciseness, and formalization of language to ensure a high standard of academic quality. It is important to emphasize that the methods, analysis, and results discussed in this report are solely based on my own work. ChatGPT was employed only to improve the expression and presentation of my report, not to generate new ideas or content. The prompts used during this process are provided in [Appendix C](#).

2. Context

2.1 Overview of Sign Language Recognition

Sign language recognition is a field dedicated to developing systems that can automatically interpret hand gestures, facial expressions, and body movements used in sign languages. These visual signs represent letters, words, or phrases, enabling communication for individuals with hearing and speech impairments. Sign language recognition systems aim to bridge communication gaps by translating these signs into spoken or written language, facilitating real-time interaction between speech/hearing-impaired individuals and those who do not know sign language. In this study, we specifically focus on recognizing hand gestures that represent individual letters of the alphabet, which is an essential step towards more comprehensive sign language recognition.

The evolution of sign language recognition has been greatly influenced by advancements in both machine learning (ML) and computer vision (CV). While Convolutional Neural Networks (CNNs) have become a core component of these systems, contributing significantly to the accuracy and efficiency of gesture recognition, feature engineering also plays a crucial role. By extracting meaningful features from raw image data, such as joint angles, hand contours, and spatial relationships between hand parts, machine learning models can make more accurate predictions. Feature extraction techniques can either complement or replace the raw pixel-based input used by CNNs, and they can be critical in handling diverse hand shapes, varying lighting conditions, and different backgrounds.

Sign language recognition systems are generally divided into two categories: sensor-based and vision-based.

2.1.1 Sensor-based Recognition

Sensor-based recognition systems offer high accuracy by tracking hand gestures through specialized devices such as gloves or motion sensors. These systems extract data on hand position, orientation, and velocity, which helps in accurately identifying sign language gestures. The primary advantage of sensor-based systems is their ability to provide skeletal data, which can lead to a high recognition rate. However, these systems require specialized equipment, can limit movement freedom, and may be costly, making them less practical for widespread use.

2.1.2 Vision-based Recognition

The shift towards vision-based approaches has opened up new possibilities for real-time sign language recognition using widely accessible devices, such as smartphones, webcams, and depth cameras. These systems use computer vision algorithms to process visual inputs, extracting key features from hand gestures and mapping them to corresponding sign language symbols or words. Vision-based systems are more flexible and do not require specialized equipment, making them suitable for use in a variety of everyday environments.

The use of deep learning techniques, particularly Convolutional Neural Networks (CNNs) and other machine learning models, has significantly improved the accuracy and efficiency of vision-based systems. These advances have enabled vision-based recognition to operate in real-time, with greater robustness in diverse lighting conditions and user environments. This has led to significant improvements in the practical applicability of sign language recognition systems, allowing for more accurate and reliable interactions in real-world settings.

Despite progress, challenges remain, such as hand occlusion, varying hand shapes, and environmental factors like lighting and background. Ongoing advancements are focused on developing systems that provide accurate and reliable real-time recognition in diverse environments, improving communication

for the deaf, hard-of-hearing, and mute communities. These advancements involve not only enhancing CNN architectures but also refining feature engineering techniques to further boost the performance of vision-based systems.

This section presents the current state of knowledge on vision-based sign language recognition, specifically focusing on American Sign Language (ASL). It reviews existing approaches, their strengths, limitations, and potential directions for future improvement. Based on the research papers I have studied, I discuss their approaches and findings, providing insights into various methods, techniques, and models for sign language recognition. These studies have informed the approach taken in this project, helping to shape the development of the proposed solution by highlighting the strengths and challenges of current systems.

2.2 Related Work

2.2.1 Shin et al. - American Sign Language Alphabet Recognition by Extracting Feature from Hand Pose Estimation

In recent advancements in American Sign Language (ASL) recognition, Shin et al. proposed a novel approach that leverages hand pose estimation for effective classification of ASL characters. The authors utilized the MediaPipe hands algorithm to extract joint coordinates from RGB images, which allowed for the generation of two distinct types of features: distance-based features and angle-based features. Distance-based features focus on the spatial relationships between hand joints, providing a clear representation of hand shape. Angle-based features, on the other hand, capture the orientation of these joints in relation to the 3D axes, which is crucial for distinguishing between similar hand signs that may differ in inclination.

To optimize the performance of their classifiers—Support Vector Machine (SVM) and Light Gradient Boosting Machine (Light GBM)—the authors employed grid search for hyperparameter tuning. This method involved

systematically exploring a predefined set of hyperparameter values to identify the optimal configuration for each classifier. The results showed that the classifiers achieved impressive accuracy rates of 87.60% for SVM and 86.12% for Light GBM on the ASL Alphabet dataset, thereby outperforming previous methodologies.

This research not only highlights the effectiveness of vision-based techniques in ASL recognition but also suggests potential applications for recognizing sign languages from other cultures and air writing, contributing to the broader field of communication technologies for the deaf and mute community.

In this project, the methodology proposed by Shin et al. served as an inspiration for developing Model 2 and Model 3, as detailed in Section 3.4. These models incorporate distance-based and angle-based features derived from hand pose estimation but further enhance their representation and extraction methods to improve classification accuracy and generalizability.

2.2.2 Kołodziej et al. - Using Deep Learning for Real-Time ASL Recognition

Kołodziej et al. (2022) focused on a deep learning-based vision system for real-time recognition of 24 static ASL signs. They employed the DenseNet architecture, trained on a combination of publicly available ASL datasets, achieving an accuracy of 80.7%. This study emphasized the significance of environmental factors such as hand coverage area and lighting conditions, which can impact the performance of recognition systems. The authors also conducted a comparative analysis between DenseNet and conventional Convolutional Neural Networks (CNNs), demonstrating the superior performance of DenseNet in handling complex hand gestures and improving recognition accuracy in diverse settings.

2.2.3 Singh et al. - Enhancing Sign Language Detection through Mediapipe and Convolutional Neural Networks (CNN)

Singh et al. (2023) proposed an innovative approach combining MediaPipe for hand tracking with Convolutional Neural Networks (CNN) for gesture classification, achieving a remarkable accuracy of 99.12% on ASL datasets . By leveraging MediaPipe's precise hand tracking and CNN's efficient feature extraction capabilities, the system demonstrated significant improvements in both speed and accuracy. The high accuracy achieved demonstrates the practical feasibility of such models for real-time applications. Their approach aligns with the goals of this project, suggesting that integrating hand tracking with CNN could be an effective method for enhancing accuracy and efficiency in real-world scenarios.

2.2.4 Adeyanju et al. - Development of an American Sign Language Recognition System using Canny Edge and Histogram of Oriented Gradient

The study by Adeyanju et al. (2021) employed Canny edge detection and Histogram of Oriented Gradients (HOG) for feature extraction, followed by K-Nearest Neighbor (K-NN) classification . Their ASL recognition system achieved an impressive accuracy of 99% on the Kaggle ASL alphabet dataset, surpassing the performance of many existing models. This work underscores the advantages of combining traditional image processing techniques with machine learning classifiers, providing insights into alternative feature extraction methods that could benefit this project.

2.3 Implications for the Current Project

The studies reviewed in this chapter highlight a variety of approaches and algorithms that address key challenges in sign language recognition, such as accuracy, real-time performance, and adaptability to environmental conditions. The findings from Shin et al. (2021) suggest that angle-based features, derived from hand pose estimation, can significantly enhance classification performance.

By capturing the angles between various hand joints, these features improve the system's ability to differentiate between similar hand signs, a technique that will be considered in this project. Additionally, Singh et al. (2023) demonstrated that integrating pose estimation and hand tracking can further enhance the accuracy of gesture classification.

The insights from Adeyanju et al. (2021) regarding HOG and edge detection provide valuable reference points for alternative feature extraction techniques, which could improve model robustness. While Kołodziej et al. (2022) and Singh et al. (2023) highlight the potential of deep learning models for gesture recognition, this project will focus on engineering features that can significantly enhance model performance without relying on computationally expensive deep learning architectures. By focusing on feature engineering techniques, such as distance-based and angle-based features, this project seeks to design a system that achieves high accuracy while maintaining computational efficiency.

Rather than employing Convolutional Neural Networks (CNNs), the focus will be on experimenting with machine learning models such as Support Vector Machines (SVMs), as these models align with the project's objectives of balancing accuracy and efficiency. The emphasis on lightweight, interpretable models ensures that the system remains accessible and deployable on devices with limited computational resources.

This project builds on these foundational studies by integrating robust feature extraction techniques with efficient machine learning methods to evaluate their effectiveness in recognizing ASL gestures. By doing so, it aims to contribute to the development of an accessible and accurate sign language recognition system capable of facilitating communication for the deaf and mute community.

3. Methods

In this chapter, the methods used to conduct the research are described in detail. The methods cover various aspects, including data gathering, data preprocessing, feature extraction, model training, and evaluation. The goal is to provide a comprehensive understanding of the procedures followed to achieve the project's objectives and enable the replication of the work. The methods chosen are justified based on the project's requirements, and pertinent literature has been referenced to inform the decisions made.

The work was developed using Python 3.12.7 within a Jupyter Notebook environment on a macOS Sonoma 14.3.1 system with an Apple M1 chip, 8GB of RAM, and a 245.11 GB SSD. The project utilized an Anaconda environment for package management. The following Python libraries and their respective versions were used:

- os: posix
- tqdm: 4.66.5
- cv2: 4.10.0
- matplotlib: 3.9.2
- torch: 2.5.1
- torchvision: 0.20.1
- numpy: 1.26.4
- skimage: 0.24.0
- sklearn: 1.5.1
- mediapipe: 0.10.18

3.1 Dataset Description

The dataset used in this project consists of images from the American Sign Language (ASL) alphabet [\[6\]](#), sourced from Kaggle. The images are organized into 29 classes, with 26 classes representing the letters A-Z and 3 classes for SPACE, DELETE, and NOTHING. The images are stored in folders corresponding to these classes, with each image having dimensions of 200x200 pixels.

This dataset provided a comprehensive and balanced representation of ASL alphabet gestures, making it suitable for training machine learning models aimed at gesture recognition. Figure 1 shows sample images of each alphabet sign in ASL.



Figure 1: Sample Images Representing Each Alphabet Sign in ASL

3.2 Data Preprocessing

To ensure consistency and improve model performance, several preprocessing steps were applied to the dataset. First, all images were resized to 128x128 pixels to standardize the input dimensions for the models. Given that the dataset predominantly contains left-hand gestures, data augmentation was applied through random horizontal flips. This transformation is crucial to ensure that the model can generalize well to both left- and right-hand gestures, thereby improving recognition accuracy for various hand orientations.

The images were also converted into tensor format, which is required for input into machine learning models. These preprocessing steps were implemented using the PyTorch library's transforms module.

3.3 Feature Extraction Methods

Feature extraction plays a critical role in converting raw hand gesture images into meaningful representations that can be processed by machine learning models. In this project, I used MediaPipe's hand pose estimation model to extract key features of hand gestures, focusing on geometric properties of the hand such as finger angles, inter-finger angles, finger curl, and palm orientation. These features are derived from the 3D keypoints detected by MediaPipe, which represents each finger joint and the palm's position in space.

In addition, Histogram of Oriented Gradients (HOG) was used as a feature extraction method in the base model to encode shape and texture information from the hand gesture images. This provided a baseline for evaluating the performance of more advanced feature extraction methods used in subsequent models.

3.3.1 Histogram of Oriented Gradients (HOG)

The Histogram of Oriented Gradients (HOG) is a popular feature extraction technique used primarily in image processing. It focuses on capturing edge information, which is essential for recognizing shapes and patterns in images. HOG works by calculating the gradient of pixel intensity at each point in the image and then grouping these gradients into a histogram based on the direction (orientation) of the edges. These histograms are then used as feature descriptors.

Calculation of HOG Features

The process of extracting HOG features involves several key steps, as implemented in the code:

1. **Grayscale Conversion:** The input RGB image is converted to grayscale using the formula:

$$I_{gray}(x, y) = 0.299 \cdot R(x, y) + 0.587 \cdot G(x, y) + 0.114 \cdot B(x, y)$$

Where R, G and B are the red, green, and blue colour channels, respectively.

2. **Image Resizing:** The grayscale image is resized to 128×64 pixels, as this is an optimal size for HOG. Resizing ensures consistent feature dimensions across all images.
3. **Gradient Computation:** The gradient of the image intensity is computed for each pixel.

The gradient is defined as:

$$G_x(x, y) = I(x + 1, y) - I(x - 1, y), G_y(x, y) = I(x, y + 1) - I(x, y - 1)$$

Where G_x and G_y are the gradients along the horizontal and vertical directions.

The magnitude and orientation of the gradient at each pixel are calculated as:

$$\text{Magnitude: } M(x, y) = \sqrt{G_x(x, y)^2 + G_y(x, y)^2}$$

$$\text{Orientation: } \theta(x, y) = \arctan \frac{G_y(x, y)}{G_x(x, y)}$$

4. Histogram Binning: The image is divided into cells of size 16×16 pixels. Within each cell, the gradient orientations are grouped into 8 bins (corresponding to 8 equally spaced angles, 0° to 180°). The magnitude of the gradient contributes to the histogram bin corresponding to its orientation.
5. Block Normalization: To achieve illumination invariance, cells are grouped into overlapping blocks of size 2×2 cells, and the histograms within each block are normalized. The normalization is performed using:

$$\text{L2-Norm: } v' = \frac{v}{\sqrt{\|v\|_2^2 + \epsilon^2}}$$

Where v is the concatenated histogram vector of a block, $\|v\|_2$ is its Euclidean norm, and ϵ is a small constant to avoid division by zero.

6. Feature Vector Construction: The normalized histograms from all blocks are concatenated into a single feature vector f , which serves as the HOG descriptor.

Implementation Details

- Grayscale Conversion: Performed using `color.rgb2gray` function.
- Resizing: The image is resized to 128×64 using `resize` function.
- Gradient and Histogram Calculation: These steps are carried out internally by the `hog` function from the `skimage.feature` module, with parameters:
 - `orientations=8`: Number of orientation bins.
 - `pixels_per_cell=(16, 16)`: Size of each cell.
 - `cells_per_block=(2, 2)`: Number of cells per block.
 - `visualize=True`: Enables visualization of the HOG image.

Feature Representation

The final HOG feature vector for each image has a length determined by the number of blocks and the number of bins per block. For the 128×64 image size with the given parameters, the length of the HOG feature vector is:

$$\text{Feature Length} = (\text{number of blocks}) \times (\text{cells per block})^2 \times \text{number of bins}$$

In this project, HOG descriptors capture the essential shape and edge details of ASL gestures, providing a robust feature set for distinguishing different hand gestures.

3.3.2 Hand Landmark Extraction

Hand landmark extraction is an essential step in capturing the positions of key joints in the hand, which are critical for recognizing gestures in ASL. MediaPipe's hand pose estimation model provides 21 keypoints per hand, corresponding to the wrist, the base joints (Metacarpophalangeal or MCP joints) of each finger, the middle joints (Proximal Interphalangeal or PIP joints), and the tip joints of the fingers. The MCP joint is where the finger connects to the hand, while the Proximal Interphalangeal (PIP) joint is the middle joint of each finger. The tip joint refers to the farthest joint of each finger, at the fingertip. These keypoints are detected in 3D space, with each keypoint represented by its x, y, and z coordinates. Figure 2 visualises the keypoints [12].

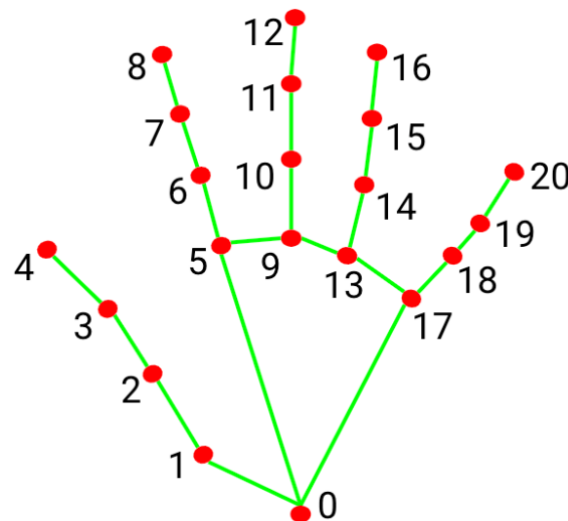


Figure 2: Mediapipe Hand Keypoints (Google AI, 2024)

Keypoints and Coordinate System

The keypoints used in this project are as follows:

- Keypoint 0: Wrist

- Keypoints 1-4: Thumb (MCP to tip)
- Keypoints 5-8: Index finger (MCP to tip)
- Keypoints 9-12: Middle finger (MCP to tip)
- Keypoints 13-16: Ring finger (MCP to tip)
- Keypoints 17-20: Pinky finger (MCP to tip)

Each keypoint has three components: x (horizontal position), y (vertical position), and z (depth). The x and y values are normalized between 0 and 1, which means they are relative to the image's width and height. The z value represents the depth of the keypoint relative to the camera, with larger values indicating that the point is farther from the camera. Figure 3 below displays the hand landmarks as detected on a webcam input, illustrating the keypoints used for feature extraction.



Figure 3: Hand Landmarks Detected on Webcam Input

Conversion to Pixel Coordinates

Since MediaPipe's keypoints are normalized to the range [0, 1], they need to be converted to actual pixel coordinates in the image. The x and y values are scaled to the image's dimensions by multiplying them by the image's width and height, respectively. This process ensures that the keypoints are represented in pixel coordinates, making them directly usable for image processing and feature extraction tasks.

For instance, the wrist (keypoint 0) would have pixel coordinates calculated as:

$$x_0 = x * \text{image_width}$$

$$y_0 = y * \text{image_height}$$

Where x and y are the normalized values, and $image_width$ and $image_height$ are the dimensions of the input image. The z value is retained as it represents the depth (distance from the camera), which is crucial for subsequent calculations, such as determining finger curl and palm orientation.

Importance of Keypoints for Feature Extraction

These keypoints are then used to compute a variety of features essential for hand gesture recognition. For example:

- **Finger Angles:** The angles between consecutive joints of each finger provide information about the flexion of the fingers.
- **Inter-Finger Angles:** These angles measure the relative orientation of adjacent fingers.
- **Finger Curl:** The degree of finger curl is assessed by measuring distances between specific joints on the finger.
- **Palm Orientation:** The rotation of the palm is derived by calculating vectors from the wrist to the base joints of the fingers and computing the normal vector of the palm's plane.

By extracting the positions of these keypoints and converting them into meaningful spatial features, the system is able to capture detailed geometric information about the hand's posture, which is crucial for accurate gesture recognition.

3.3.3 Distance Features

Distance-based features provide valuable information about the spatial relationships between key points on the hand. These features help capture the overall shape and position of the hand, which is important for distinguishing different gestures. In this implementation, the distances between the wrist and each fingertip are computed, as these distances reflect the spread and position of the fingers relative to the wrist.

Keypoints: The keypoints used for computing distances are the wrist (keypoint 0) and the tips of each finger:

- | | |
|--------------------------------|----------------------------------|
| • Keypoint 0: Wrist | • Keypoint 12: Middle finger tip |
| • Keypoint 4: Thumb tip | • Keypoint 16: Ring finger tip |
| • Keypoint 8: Index finger tip | • Keypoint 20: Pinky finger tip |

Calculation: The Euclidean distance formula is used to calculate the straight-line distance between the wrist (keypoint 0) and each fingertip. Specifically, the distance between two keypoints is calculated as:

$$distance = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

Where (x_1, y_1, z_1) and (x_2, y_2, z_2) are the coordinates of the two points. In this case, the wrist (keypoint 0) is compared with each fingertip (keypoints 4, 8, 12, 16, and 20). Specifically, the distance between the wrist and each fingertip is calculated as follows:

- Wrist to Thumb Tip
- Wrist to Index Tip
- Wrist to Middle Tip
- Wrist to Ring Tip
- Wrist to Pinky Tip

These distances are computed for each fingertip, providing a set of features that reflect how far each fingertip is from the wrist. This information is useful for recognizing hand gestures based on the spread of the fingers and the overall position of the hand.

In the implementation, the calculation is performed using NumPy's `np.linalg.norm()` function. This function computes the norm (or magnitude) of a vector, which in this case represents the difference between two points in 3D space. The use of `np.linalg.norm()` simplifies distance calculations by internally handling the mathematical operations for vector magnitude. It ensures high computational efficiency and reduces the risk of manual calculation errors, making it a robust choice for extracting distance-based features.

3.3.4 Joint Angle Features

The joint angle features represent the angles at the mid joints of each finger, which help capture the relative positioning of the finger segments. They provide a detailed representation of the relative positioning and curvature of fingers, which is essential for distinguishing various sign language gestures. These features are calculated by analysing the angles formed at specific hand joints using a Python function designed for this purpose.

Python Function Implementation

A Python function, `calculate_angle` was created to compute the angle at a joint (B) formed by two connected vectors (BA and BC). The inputs to the function are three points:

A: The first point forming the first vector.

B: The central joint where the angle is formed.

C: The third point forming the second vector.

Functionality:

1. Compute the vectors:

$$\overrightarrow{BA} = A - B, \quad \overrightarrow{BC} = C - B$$

2. Calculate the dot product of the two vectors:

$$\text{Dot Product} = \overrightarrow{BA} \cdot \overrightarrow{BC}$$

3. Determine the magnitudes (norms) of the vectors:

$$\|BA\| = \sqrt{\text{sum of squares of } \overrightarrow{BA}}, \quad \|BC\| = \sqrt{\text{sum of squares of } \overrightarrow{BC}}$$

4. Compute the cosine of the angle:

$$\cos(\theta) = \frac{\text{Dot Product}}{\|BA\| \cdot \|BC\|}$$

To handle numerical precision issues, the cosine value is clipped to the range $[-1,1]$.

5. Convert the angle from radians to degrees:

$$\theta = \arccos(\cos(\theta)) \times \frac{180}{\pi}$$

This function outputs the angle in degrees for easy interpretability.

Joint Angle Computations

Using the `calculate_angle` function, angles are calculated for critical joints across all fingers:

1. Index Finger:

- MCP (metacarpophalangeal) joint: Angle at Keypoints [5,6,7].

- PIP (proximal interphalangeal) joint: Angle at Keypoints [6,7,8].
2. Middle Finger:
 - MCP joint: Angle at Keypoints [9,10,11].
 - PIP joint: Angle at Keypoints [10,11,12].
 3. Ring Finger:
 - MCP joint: Angle at Keypoints [13,14,15].
 - PIP joint: Angle at Keypoints [14,15,16].
 4. Pinky Finger:
 - MCP joint: Angle at Keypoints [17,18,19].
 - PIP joint: Angle at Keypoints [18,19,20].
 5. Thumb:
 - Base joint: Angle at Keypoints [0,1,2].
 - Tip joint: Angle at Keypoints [1,2,4].

Advantages of Joint Angle Features

- Invariant to Scale: Angles remain consistent regardless of variations in hand size or image resolution.
- High Discriminative Power: These features capture the intricacies of finger curvatures and joint positions, which are vital for distinguishing between sign gestures.

3.3.5 Inter-Finger Angles

Inter-finger angles refer to the angles formed between the vectors corresponding to the fingers of the hand. These angles provide important information about the relative positioning of the fingers, which can be useful in distinguishing different hand gestures. In this method, we compute the angles formed by the vectors between key points across different fingers.

To calculate the inter-finger angles, I used the same `calculate_angle` function previously mentioned in section 3.3.4, which calculates the angle at a given point based on the vectors formed by its neighbouring points. For the inter-finger angles, the function is applied between specific key points that represent the base joints of adjacent fingers.

The following calculations are performed to obtain the inter-finger angles:

- Thumb to Index Finger: The angle between the vectors formed by the thumb's base joint (keypoint 1), the index finger's base joint (keypoint 5), and the index finger's next joint (keypoint 9).

- Index to Middle Finger: The angle between the vectors formed by the index finger's base joint (keypoint 5), the middle finger's base joint (keypoint 9), and the middle finger's next joint (keypoint 13).
- Middle to Ring Finger: The angle between the vectors formed by the middle finger's base joint (keypoint 9), the ring finger's base joint (keypoint 13), and the ring finger's next joint (keypoint 17).

The angles are calculated using the dot product formula to determine the cosine of the angle between the two vectors and then applying the inverse cosine function to obtain the angle in radians. The angle is then converted from radians to degrees. The resulting inter-finger angles provide a useful set of features to distinguish different hand gestures, especially when fingers are spread or positioned in specific ways.

3.3.6 Finger Curl

In addition to geometric features based on distances and angles, finger curl provides another useful representation of hand posture. Finger curl refers to the amount of bending in a finger, typically measured as the ratio of the distance from the base joint (Metacarpophalangeal or MCP) to the tip of the finger, divided by the distance from the base joint to the Proximal Interphalangeal (PIP) joint.

The mathematical formulation for calculating the curl of a finger is:

$$Curl = \frac{\text{Distance from MCP to Tip}}{\text{Distance from MCP to PIP}}$$

Where:

- The MCP is the base joint of the finger.
- The PIP is the second joint in the finger, located between the MCP and the finger-tip.
- The tip is the distal endpoint of the finger.

To calculate the finger curl for each finger, a Python function called `calculate_curl()` has been created. This function takes as input three key points: the MCP joint (base), the PIP joint, and the tip of the finger. The function performs the following steps:

- Computes the distance between the MCP joint and the finger-tip.
- Computes the distance between the MCP joint and the PIP joint.

- Calculates the ratio of the distance from the MCP to the tip, and the distance from the MCP to the PIP.

This ratio represents the curl of the finger. If the curl ratio is close to 1, the finger is relatively straight. A larger value indicates that the finger is more curled.

The `calculate_curl()` function is applied to each of the five fingers (thumb, index, middle, ring, and pinky), and the resulting curl values are stored in a list. These curl values provide additional information about the hand's gesture, helping to differentiate between various hand shapes.

3.3.7 Palm Orientation

Palm orientation describes the rotation of the palm, which is crucial for recognizing gestures where the orientation of the palm matters, such as in signs for "I" and "J." These two signs are visually similar, with the only distinguishing factor being the orientation of the palm. Therefore, accurately capturing palm orientation is essential for differentiating between such signs.

To calculate the palm orientation, the positions of three key landmarks are used: the wrist, the middle finger's metacarpophalangeal (MCP) joint, and the pinky finger's MCP joint. The palm's orientation is then derived in terms of three rotational angles: roll, pitch, and yaw.

- Roll refers to the rotation of the palm around the axis that runs from the wrist to the fingertips, often described as tilting left or right.
- Pitch is the rotation around the axis that runs from the thumb to the pinky, representing the up or down tilt of the palm.
- Yaw describes the rotation around the vertical axis, controlling the left or right turning of the palm.

A Python function, `calculate_palm_orientation` is created to compute these angles as follows:

- Vector Calculation: The wrist-to-middle and wrist-to-pinky vectors are obtained by subtracting the wrist's coordinates from the coordinates of the middle and pinky MCP joints, respectively. These vectors represent the directional alignment of the palm in space.
- Palm Normal Vector: The normal vector to the palm plane is computed by taking the cross product of the wrist-to-middle and wrist-to-pinky vectors.

This normal vector is then normalized to unit length to eliminate the effects of scale.

- Rotation Angles:
 - Roll is calculated using the arctangent of the ratio of the y-component to the z-component of the palm normal vector.
 - Pitch is computed by finding the arctangent of the negative x-component of the palm normal vector, divided by the magnitude of its y and z components.
 - Yaw is determined as the arctangent of the ratio of the y-component of the wrist-to-middle vector to the x-component.

These angles are converted from radians to degrees for more intuitive interpretation.

By analysing the palm's orientation, this feature helps distinguish between gestures that are visually similar but differ based on how the palm is rotated, ensuring accurate recognition in gesture-based systems.

Conclusion of Feature Extraction

After extracting the individual hand features, such as finger angles, inter-finger angles, finger curls, and palm orientation, the next step was to combine these features into a single feature vector for each hand gesture. This final feature vector serves as the input to the machine learning model.

3.3 Data Splitting

Data splitting is a crucial step in machine learning to ensure that the model is trained on one portion of the data and evaluated on another, unseen portion. In this study, the dataset was divided into training and testing sets using the `train_test_split` function from the `sklearn.model_selection` library.

For each feature set, and the corresponding labels, an 80-20 split was applied, where 80% of the data was used for training and 20% was reserved for testing. This division was performed using stratified sampling, which ensures that the distribution of labels is maintained across both the training and testing sets. The random state was fixed to 42 to ensure reproducibility of the results. This split allows for an effective evaluation of the model's performance on unseen data, helping to prevent overfitting and ensure generalization to real-world sign language gestures.

3.4 Model Training

In this project, I experimented with 5 different models, each utilizing different combinations of extracted features to train a Support Vector Machine (SVM). The goal was to compare the performance of these models based on the distinct sets of hand gesture characteristics they represented. Each model was trained on the features extracted from the ASL Alphabet Dataset, and the results were evaluated to determine which feature combination provided the best performance for recognizing sign language gestures.

Model 1: Baseline Model (HOG Features + SVM)

The first model serves as a baseline and uses only Histogram of Oriented Gradients (HOG) features to classify the gestures. HOG is a feature descriptor commonly used for object detection, and in this case, it captures the edge information of the hand shape. The model was trained using the Support Vector Machine (SVM), a powerful classification algorithm that works well for high-dimensional feature spaces. This model primarily focuses on the global shape and structure of the hand, providing a simple yet effective feature set for gesture recognition.

Model 2: Finger Angles + Inter-Finger Angles + SVM

The second model uses only angle-based features, specifically finger angles and inter-finger angles. Finger angles are derived from the keypoints of each finger joint, while inter-finger angles describe the relative angles between different fingers. This combination was inspired by the findings of Shin et al. (2021) [\[1\]](#), where it was stated that "*the performance increased when angle-based features are used*" (Shin et al., 2021, pp. 12).

However, it is important to note that the angle features used in this project differ from those in Shin et al.'s work. Shin et al.'s angle-based features are derived from the angles between the direction vectors of joint pairs and the X, Y, and Z axes. These angles represent the hand's orientation in 3D space:

- X-axis angle: Captures side-to-side tilt.
- Y-axis angle: Represents vertical tilt.
- Z-axis angle: Indicates forward or backward orientation.

Together, these features describe the hand's pose for gesture recognition, capturing how the joints are positioned relative to each other and reflecting the overall shape and configuration of the hand.

In contrast, the features used in this project were specifically designed to improve real-time testing performance by providing more discriminative and relevant information. For instance, instead of relying on orientation angles with respect to coordinate axes, I focused on angles formed at specific hand joints, which better reflect the nuances of sign language gestures. These angle features, along with complementary features such as finger curls, inter-finger angles, and palm orientation (introduced in later models), were chosen to enhance the model's ability to generalize to real-world scenarios.

The SVM classifier was trained using these angle-based features, providing a robust foundation for comparing gesture recognition performance.

Model 3: Distance Features + Finger Angles + Inter-Finger Angles + SVM

The third model combines distance-based features with the finger angles and inter-finger angles used in Model 2. Distance features capture the spatial relationships between the wrist and the fingertips, providing critical information about the hand's spread and overall configuration. By combining these features with the angle-based features, the model can better distinguish between various hand gestures.

This combination was also inspired by the findings of Shin et al. (2021) [\[1\]](#), who suggested that a combination of distance-based and angle-based features derived from hand pose estimation can significantly enhance classification performance. However, the distance features used in Shin et al.'s work differ from those in this project. In their work, distance features are extracted from the 21 hand joint coordinates by calculating the Euclidean distances between all possible pairs of joints, excluding neighbouring joints (e.g., joints directly connected by bones). This approach provides a dense feature set that emphasizes spatial relationships between non-adjacent joints.

In contrast, this project uses more targeted distance features designed for real-time testing scenarios. Specifically, the distances between the wrist and each fingertip were selected to provide more informative and computationally efficient features. Additionally, the angle features used in this model are the same as those in Model 2, focusing on angles formed at specific hand joints to capture nuanced hand gestures.

The inclusion of both distance and angle features allows this model to capture both local (finger-specific) and global (hand-wide) characteristics, resulting in a richer feature set for training the SVM classifier.

While Shin et al. reported an accuracy of 87.06% on their test set using their angle features, the goal of this project is to achieve not only high test accuracy but also strong performance in real-time gesture recognition.

Model 4: Finger Angles + Inter-Finger Angles + Finger Curls + SVM

The third model focuses on more intricate finger-related features, including finger angles, inter-finger angles, and finger curls. Finger curls quantify how much each finger is bent, adding another layer of detail to the hand's pose. This model benefits from the additional flexibility of finger curl features, which capture dynamic hand movements more effectively than static hand shapes alone. By combining these features with finger angles and inter-finger angles, the model can better recognize hand gestures that involve varying degrees of finger flexion, such as for specific sign language letters or actions. The SVM classifier was again employed for training.

Model 5: Finger Angles + Inter-Finger Angles + Finger Curls + Palm Orientation + SVM

The final model incorporates the most comprehensive set of features, including finger angles, inter-finger angles, finger curls, and palm orientation. Palm orientation is an important feature for sign language recognition, as it describes the orientation of the palm, which can significantly affect the meaning of a gesture. By including palm orientation along with the other features, this model can capture a broader range of hand gestures and their variations, leading to more accurate predictions. Like the previous models, SVM was used to train the classifier.

Training Overview

Each model was trained on the features extracted from the dataset, and the SVM was configured with a polynomial kernel, which is commonly used in situations where the feature space is non-linearly separable. The polynomial kernel allows the SVM to capture more complex relationships between the features and the target classes. By comparing the performance of these models, I aimed to determine the most effective combination of features for sign language gesture recognition using the ASL alphabet.

3.5 Model Evaluation

The trained models were evaluated using a combination of quantitative metrics and visualization techniques to assess their performance on the test dataset. Key metrics such as accuracy, precision, recall, and F1-score were computed, providing a detailed view of each model's ability to classify ASL gestures. Additionally, a confusion matrix was used to visualize the distribution of predictions across the 29 gesture classes, offering deeper insights into the models' behaviour.

Accuracy and Classification Report

Accuracy measures the proportion of correctly classified samples out of the total samples, serving as a high-level indicator of the model's overall performance. While accuracy is a useful starting point, it is important to interpret it in conjunction with other metrics for a more thorough evaluation of the model's capabilities.

The classification report, generated using the `classification_report` function from the `sklearn.metrics` library, includes:

- **Precision:** The ratio of true positive predictions to the total positive predictions (true positives + false positives). Precision indicates how reliable the model's positive predictions are. A high precision score means that when the model predicts a class, it is likely correct.
- **Recall:** The ratio of true positive predictions to the actual number of positive samples (true positives + false negatives). Recall measures the model's ability to correctly identify all instances of a class. High recall ensures that most true gestures are detected, even at the cost of some false positives.
- **F1-Score:** The harmonic mean of precision and recall, balancing these two metrics. A high F1-score reflects a model that performs well in both identifying true positives and avoiding false positives, making it particularly useful when precision and recall need to be equally prioritized.

These metrics were calculated and displayed for each class, offering insights into the performance of the model across different ASL gestures.

Confusion Matrix Visualization

The confusion matrix provides a granular view of the model's predictions by mapping the true labels against the predicted labels. Each cell in the matrix

represents the count of predictions for a specific pair of actual and predicted classes. Diagonal entries correspond to correct predictions, while off-diagonal entries indicate misclassifications.

The matrix was computed using the `confusion_matrix` function from the `sklearn.metrics` library and visualized using the `ConfusionMatrixDisplay` class. The heatmap was created with `matplotlib`'s plotting functions, and the x-axis and y-axis labels were rotated for better readability. This visualization helps identify patterns of misclassification, such as confusion between gestures with similar shapes or orientations.

Insights from Evaluation Metrics

The evaluation process revealed:

- **High Accuracy:** Demonstrates the overall effectiveness of the models but must be interpreted alongside other metrics to account for model performance in more detail.
- **Precision and Recall Trade-offs:** High precision for a class suggests that the model rarely misclassifies other gestures as that class, while high recall indicates the model's ability to identify most instances of the gesture, even if some predictions are incorrect.
- **F1-Score Analysis:** A balanced F1-score across classes highlights the model's consistency in maintaining both precision and recall.
- **Confusion Matrix Insights:** Specific misclassification patterns observed in the matrix can guide feature engineering and model improvement efforts by highlighting gestures that require additional discriminative features.

Through this comprehensive evaluation, the strengths and weaknesses of each model were identified, enabling a thorough comparison of the feature combinations and their effectiveness in recognizing ASL gestures.

3.6 Real-Time Testing

In the real-time testing phase, the system is designed to perform hand gesture recognition using a webcam. The primary objective is to capture hand gestures in real time and predict the corresponding sign language letter or gesture based on the model used.

The process begins by capturing images from the webcam, followed by hand landmark detection using MediaPipe's hand tracking model. The hand landmarks

are extracted from the captured frame, and features are computed for classification. The relevant features used during each model's training are extracted from the real-time captured image, ensuring that the appropriate features are used for classification. These features are then fed to the trained model for prediction, and the predicted class is displayed.

3.6.1 Graphical User Interface (UI)

A graphical user interface (UI) was created using Tkinter to enhance the user experience. The UI displays a background image, a title at the top, and a live feed window showing the webcam capture in real time. A bounding box is drawn around the detected hand in each frame to highlight the region of interest. The system overlays a transparent box displaying the predicted sign language gesture at the bottom of the screen. As hand landmarks are processed, real-time predictions are updated and displayed in a text box on the UI.

3.6.2 Real-Time Testing Workflow

The steps involved in the real-time testing are as follows:

- **Webcam Capture:** A video stream is initiated from the webcam to continuously capture frames. Each frame is then processed to detect the hand landmarks.
- **Hand Landmark Detection:** MediaPipe's hand landmark detection is used to identify 21 keypoints on the hand in each frame. These keypoints represent the positions of the wrist, palm, and fingers.
- **Bounding Box Drawing:** The coordinates of the bounding box surrounding the detected hand are computed based on the hand landmarks. The bounding box is drawn around the hand to highlight the detected region, making it easier to visualize the detection area.
- **Feature Extraction:** For each frame, features are extracted based on the specific model's requirements. These features are used to create a feature vector that is fed into the model for prediction.
- **Prediction:** The extracted feature vector is passed to the trained model (e.g., SVM, Random Forest) for classification. The model predicts the corresponding gesture or letter, which is displayed in real time on the UI.
- **UI Overlay:** A transparent overlay is drawn on the live feed, where the prediction result is displayed. This ensures that the user can see the prediction along with the live video feed.

- **Exit Option:** The user can press the 'q' key to exit the real-time testing window and stop the webcam capture.

This real-time testing setup enables immediate feedback and is crucial for validating the system's performance in a dynamic and interactive environment.

3.6.3 Real-Time Testing Evaluation Setup

The real-time testing setup was evaluated under various conditions to assess the robustness and generalizability of the system. These conditions included:

- **Varying Light Conditions:** The system was tested in different lighting environments, ranging from well-lit rooms to low-light situations, to evaluate how changes in lighting affected the accuracy of hand landmark detection and, consequently, the extracted features and model performance.
- **Varying Backgrounds:** Testing was also conducted with different backgrounds, including both plain and cluttered environments. These variations were important to observe how background noise or complex backgrounds might interfere with the hand landmark detection process and influence the accuracy of predictions.
- **Left and Right Hand Testing:** The system was tested with both left and right hands to evaluate how the model performs with different hand orientations and mirror-image gestures.
- **Different Hand Positions:** The model was tested with varying hand positions, such as hands held close to the camera, far away, and at different angles. These variations were included to assess how changes in hand positioning impacted the detection of hand landmarks and the resulting classification performance.

Frame Rate and Latency:

Ensuring that the real-time system provides fast and responsive predictions without noticeable lag is crucial for user experience. During testing, the system's frame rate and latency were carefully monitored to ensure that predictions were made promptly, even in dynamic testing scenarios. Maintaining a high frame rate with minimal latency is essential for smooth user interaction and to ensure that the system can keep up with real-time hand gestures.

4. Results

This chapter presents the results obtained from the models developed and evaluated in this project. The focus is on the performance of the models in both controlled test environments and real-time settings. The results for each model are presented in terms of both quantitative metrics, such as accuracy, precision, recall, and F1-score, as well as qualitative observations from real-time testing, followed by a comparison of their performance.

4.1 Model 1: HOG Features + SVM

Model 1 was trained using Histogram of Oriented Gradients (HOG) features extracted from the ASL Alphabet Dataset. The HOG feature extraction produced a feature vector of shape (87000, 672), which was used as input to the model for training. The training process took a total of 1003.85 seconds. The detailed classification report and confusion matrix for Model 1 are as follows:

	precision	recall	f1-score	support
A	1.00	0.99	0.99	600
B	1.00	1.00	1.00	600
C	1.00	1.00	1.00	600
D	1.00	1.00	1.00	600
E	0.99	0.99	0.99	600
F	1.00	1.00	1.00	600
G	1.00	1.00	1.00	600
H	1.00	1.00	1.00	600
I	1.00	1.00	1.00	600
J	1.00	1.00	1.00	600
K	1.00	1.00	1.00	600
L	1.00	1.00	1.00	600
M	1.00	1.00	1.00	600
N	1.00	1.00	1.00	600
O	1.00	1.00	1.00	600
P	1.00	1.00	1.00	600
Q	1.00	1.00	1.00	600
R	1.00	0.99	1.00	600
S	1.00	0.99	1.00	600
T	1.00	1.00	1.00	600
U	0.99	0.99	0.99	600
V	0.99	0.99	0.99	600
W	0.99	1.00	1.00	600
X	1.00	1.00	1.00	600
Y	1.00	1.00	1.00	600
Z	1.00	1.00	1.00	600
del	1.00	1.00	1.00	600
nothing	1.00	1.00	1.00	600
space	1.00	1.00	1.00	600
accuracy			1.00	17400
macro avg	1.00	1.00	1.00	17400
weighted avg	1.00	1.00	1.00	17400

Figure 4: Model 1 Classification Report

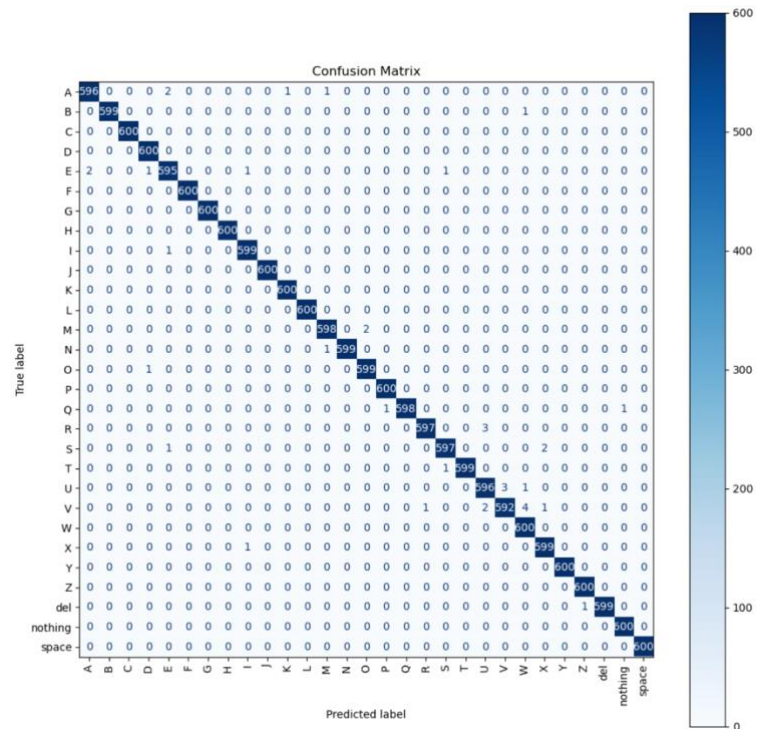


Figure 5: Model 1 Confusion Matrix

Test Set Results: The classification accuracy on the test set was found to be 99.78%. The model performed excellently across all classes, demonstrating high precision, recall, and F1-scores. The accuracy was close to 100%, reflecting the model's strong classification ability on the test set. The confusion matrix for the

model confirms that most of the alphabet classes were predicted correctly. This demonstrates the model's robust classification performance on the test set.

Real-Time Testing Performance: Despite the impressive performance on the test set, Model 1 displayed poor results when tested in real-time. The model struggled to generalize to real-world data, showing significant discrepancies between the high test accuracy and its performance in a real-time setting. This highlights a potential issue of overfitting to the test data and limited generalizability to real-time conditions.

Interestingly, some letters—C, G, H, L, and N—performed better than others in real-time testing. However, the model's overall performance in a real-time environment was much lower than expected, suggesting that it may not handle real-world variations in data such as lighting, background, or noise as effectively as it handles test set data.

4.2 Model 2: Finger Angles + Inter-Finger Angles + SVM

For Model 2, the feature extraction process involved computing Finger Angles and Inter-Finger Angles, resulting in features of shape (68739, 13). The extracted features were used to train an SVM classifier, and the training process took 731.85 seconds to complete. The detailed classification report and confusion matrix for Model 2 are as follows:

	precision	recall	f1-score	support
A	0.97	0.97	0.97	474
B	0.98	0.98	0.98	459
C	0.86	0.95	0.90	436
D	0.95	0.96	0.96	545
E	0.95	0.97	0.96	479
F	0.99	0.98	0.98	588
G	0.95	0.96	0.96	529
H	0.84	0.84	0.84	507
I	0.85	0.83	0.84	514
J	0.85	0.86	0.86	550
K	0.96	0.98	0.97	555
L	0.99	0.97	0.98	539
M	0.77	0.83	0.80	410
N	0.89	0.76	0.82	315
O	0.96	0.95	0.95	492
P	0.93	0.95	0.94	442
Q	0.95	0.87	0.91	468
R	0.82	0.81	0.81	526
S	0.97	0.95	0.96	539
T	0.99	0.97	0.98	495
U	0.81	0.81	0.81	519
V	0.95	0.95	0.95	521
W	0.98	0.96	0.97	514
X	0.96	0.94	0.95	469
Y	0.99	0.99	0.99	534
Z	0.92	0.95	0.94	500
del	0.85	0.88	0.87	412
nothing	0.00	0.00	0.00	4
space	0.84	0.84	0.84	413
accuracy			0.92	13748
macro avg	0.89	0.89	0.89	13748
weighted avg	0.92	0.92	0.92	13748

Figure 6: Model 2 Classification Report.

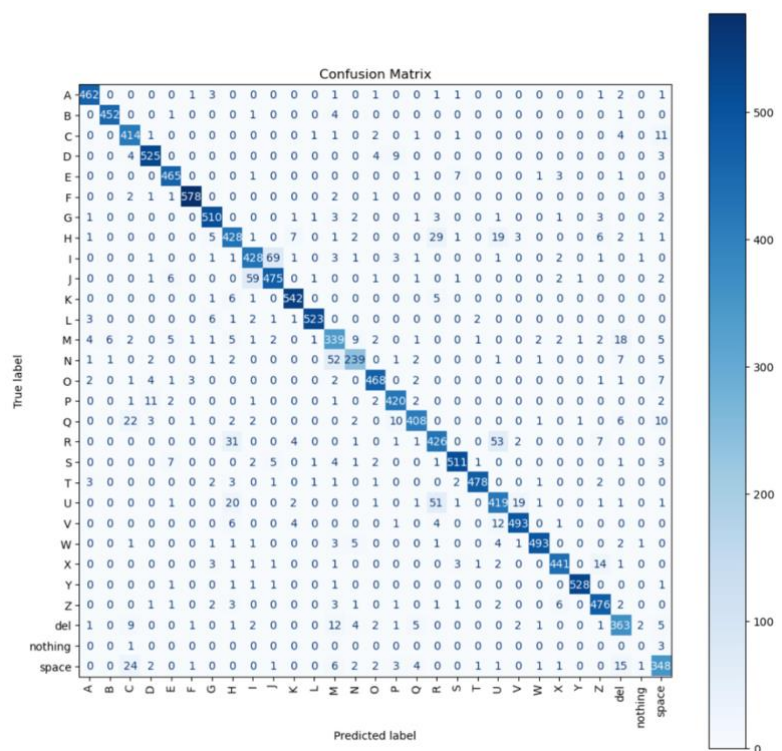


Figure 7: Model 2 Confusion Matrix

Test Set Results: The classification report for Model 2 reveals an accuracy of 92% on the test set, indicating strong performance in classifying the sign language alphabet. Precision, recall, and F1-scores for most alphabet classes were high, as shown in the classification report, demonstrating balanced performance across the different classes.

Real-Time Testing: In real-time testing, Model 2 performed well overall, correctly predicting most classes. However, it failed to classify E, I, K, R, T, U, X, and Z correctly. Despite these challenges, the real-time performance of Model 2 was better than Model 1, demonstrating moderate generalizability to real-world conditions.

4.3 Model 3: Distance Features + Finger Angles + Inter-Finger Angles + SVM

For Model 3, the feature extraction process combined distance features, finger angles, and inter-finger angles, resulting in features of shape (68,690, 18). The model was trained using an SVM classifier, and the training process took 24.4 seconds to complete. The detailed classification report and confusion matrix for Model 3 are as follows:

	precision	recall	f1-score	support
A	0.96	0.99	0.97	473
B	0.99	1.00	0.99	459
C	0.93	0.97	0.95	438
D	0.98	0.97	0.97	545
E	0.98	0.99	0.99	479
F	0.99	0.99	0.99	586
G	0.99	0.98	0.99	528
H	0.93	0.92	0.93	507
I	0.87	0.84	0.86	514
J	0.86	0.90	0.88	549
K	0.99	0.98	0.98	554
L	0.99	0.99	0.99	539
M	0.84	0.91	0.87	411
N	0.92	0.84	0.88	315
O	0.98	0.99	0.98	490
P	0.96	0.95	0.96	443
Q	0.95	0.93	0.94	469
R	0.91	0.87	0.89	525
S	0.98	0.97	0.97	539
T	0.98	0.97	0.98	493
U	0.87	0.90	0.88	518
V	0.97	0.98	0.98	522
W	0.99	0.98	0.98	512
X	0.97	0.97	0.97	470
Y	1.00	0.99	1.00	533
Z	0.98	0.97	0.97	498
del	0.92	0.95	0.94	412
nothing	1.00	0.50	0.67	4
space	0.94	0.91	0.92	413
accuracy			0.95	13738
macro avg	0.95	0.93	0.94	13738
weighted avg	0.95	0.95	0.95	13738

Figure 8: Model 3 Classification Report

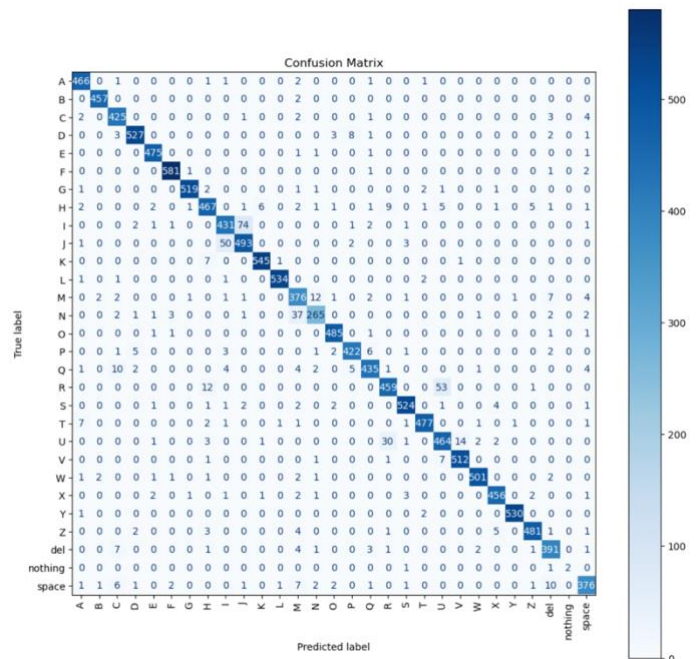


Figure 9: Model 3 Confusion Matrix

Test Set Results: The classification report for Model 3 reveals an accuracy of 95% on the test set, indicating that the model performed well in classifying the sign language alphabet. The precision, recall, and F1-scores were also

impressive, demonstrating balanced performance across the different classes. The confusion matrix for Model 3 showed that the majority of classes were predicted correctly, with only a few misclassifications in certain classes, most notably for 'nothing' and 'space,' where the accuracy was notably lower. This suggests that the model is capable of handling most alphabet signs well.

Real-Time Testing: Despite the high accuracy on the test set, Model 3 exhibited poor real-time performance. When applied to real-time testing using live webcam data, the model struggled significantly, only correctly predicting the letters S, T, and Y. This discrepancy between the test set performance and real-time results highlights that the model did not generalize well to real-world conditions, similar to the performance of Model 1.

4.4 Model 4: Finger Angles + Inter-Finger Angles + Finger Curls + SVM

For Model 4, the feature extraction process combined Finger Angles + Inter-Finger Angles + Finger Curls, resulting in features of shape (68719, 18). The extracted features were used to train an SVM classifier, and the training process took approximately 19.49 seconds. The detailed classification report and confusion matrix for Model 4 are as follows:

	precision	recall	f1-score	support
A	0.96	0.97	0.96	475
B	0.98	1.00	0.99	458
C	0.86	0.93	0.89	437
D	0.97	0.97	0.97	545
E	0.96	0.97	0.96	479
F	0.99	0.97	0.98	587
G	0.98	0.95	0.96	528
H	0.81	0.79	0.80	507
I	0.80	0.74	0.77	514
J	0.80	0.81	0.80	548
K	0.92	0.95	0.94	554
L	0.99	0.98	0.98	539
M	0.80	0.87	0.83	410
N	0.89	0.77	0.83	317
O	0.95	0.97	0.96	488
P	0.95	0.96	0.95	445
Q	0.94	0.88	0.90	472
R	0.83	0.82	0.83	525
S	0.95	0.96	0.95	539
T	0.97	0.97	0.97	495
U	0.80	0.79	0.80	517
V	0.93	0.93	0.93	521
W	0.99	0.96	0.97	512
X	0.95	0.93	0.94	468
Y	0.99	0.99	0.99	534
Z	0.91	0.94	0.92	499
del	0.77	0.91	0.84	411
nothing	1.00	0.67	0.80	3
space	0.87	0.79	0.83	417
accuracy			0.91	13744
macro avg	0.91	0.90	0.91	13744
weighted avg	0.91	0.91	0.91	13744

Figure 10: Model 4 Classification Report

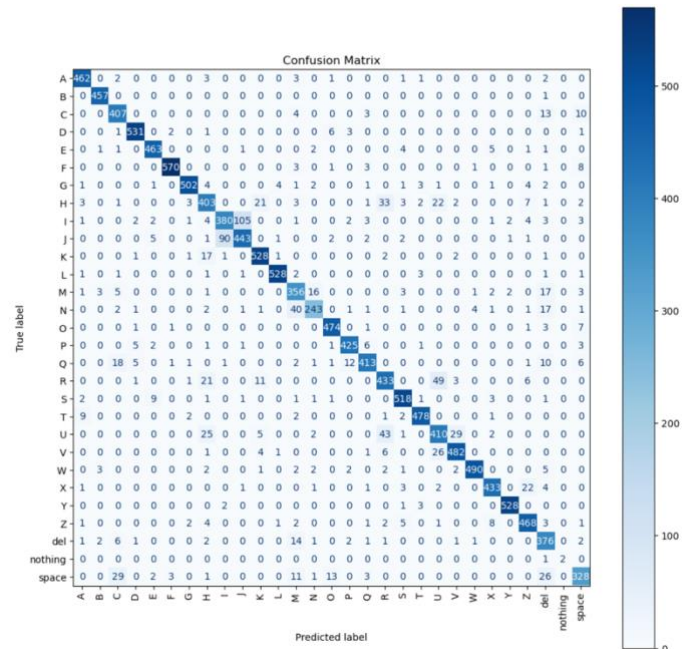


Figure 11: Model 4 Confusion Matrix

Test Set Results: The classification report yielded an overall accuracy of 91%, indicating strong performance on the test set. Precision, recall, and F1-scores for most alphabet classes were high, demonstrating effective classification for most

letters. The confusion matrix confirms that the model performed well for many alphabet classes, with some slight confusion between letters that share similar features. This performance highlights the model's ability to classify most alphabet gestures effectively on the test data.

Real-Time Testing: When tested in real-time, Model 4 showed high performance, correctly predicting most alphabet classes. However, it struggled with the letters I, R, and U, which were not classified correctly in real-world scenarios. Despite these minor challenges, the real-time performance of Model 4 was significantly better than the previous models, demonstrating improved generalizability to real-world conditions.

4.5 Model 5: Finger Angles + Inter-Finger Angles + Finger Curls + Palm Orientation + SVM

For Model 5, the feature extraction process combined Finger Angles, Inter-Finger Angles, Finger Curls, and Palm Orientation, resulting in a feature set of shape (66419, 21). The SVM model training process for this feature set was highly efficient, completing in just 9.63 seconds. . The detailed classification report and confusion matrix for Model 5 are as follows:

	precision	recall	f1-score	support
A	0.95	0.98	0.96	453
B	0.98	1.00	0.99	452
C	0.93	0.98	0.95	414
D	0.98	0.98	0.98	533
E	0.94	0.98	0.96	473
F	0.98	0.98	0.98	581
G	0.99	0.99	0.99	507
H	0.97	0.98	0.98	494
I	0.96	0.94	0.95	501
J	0.99	0.95	0.97	538
K	0.95	0.98	0.96	546
L	0.99	1.00	0.99	526
M	0.81	0.89	0.85	365
N	0.87	0.77	0.82	270
O	0.96	0.98	0.97	475
P	0.97	0.98	0.98	419
Q	0.97	0.94	0.96	452
R	0.86	0.83	0.84	518
S	0.96	0.96	0.96	526
T	0.98	0.95	0.97	487
U	0.80	0.79	0.80	508
V	0.93	0.91	0.92	516
W	0.99	0.97	0.98	506
X	0.98	0.97	0.98	450
Y	0.98	0.98	0.98	528
Z	0.97	0.95	0.96	486
del	0.92	0.96	0.94	384
nothing	0.67	0.67	0.67	3
space	0.93	0.91	0.92	373
accuracy			0.95	13284
macro avg	0.94	0.94	0.94	13284
weighted avg	0.95	0.95	0.95	13284

Figure 12: Model 5 Classification Report

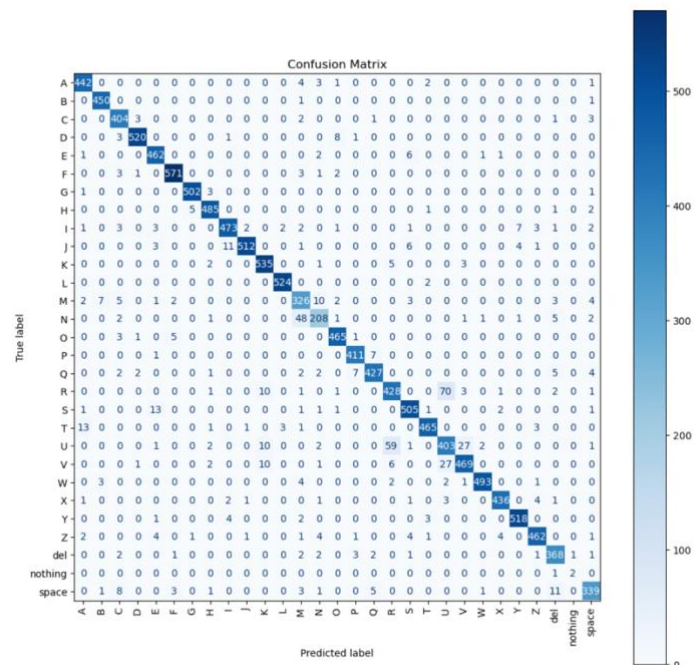


Figure 13: Model 5 Confusion Matrix

Test Set Results: The classification report for Model 5 on the test set reveals a strong overall performance, achieving an impressive accuracy of 95%. The model demonstrated high precision, recall, and F1-scores across most classes,

with relatively few misclassifications. This robust performance highlights the effectiveness of combining the four feature types for classification.

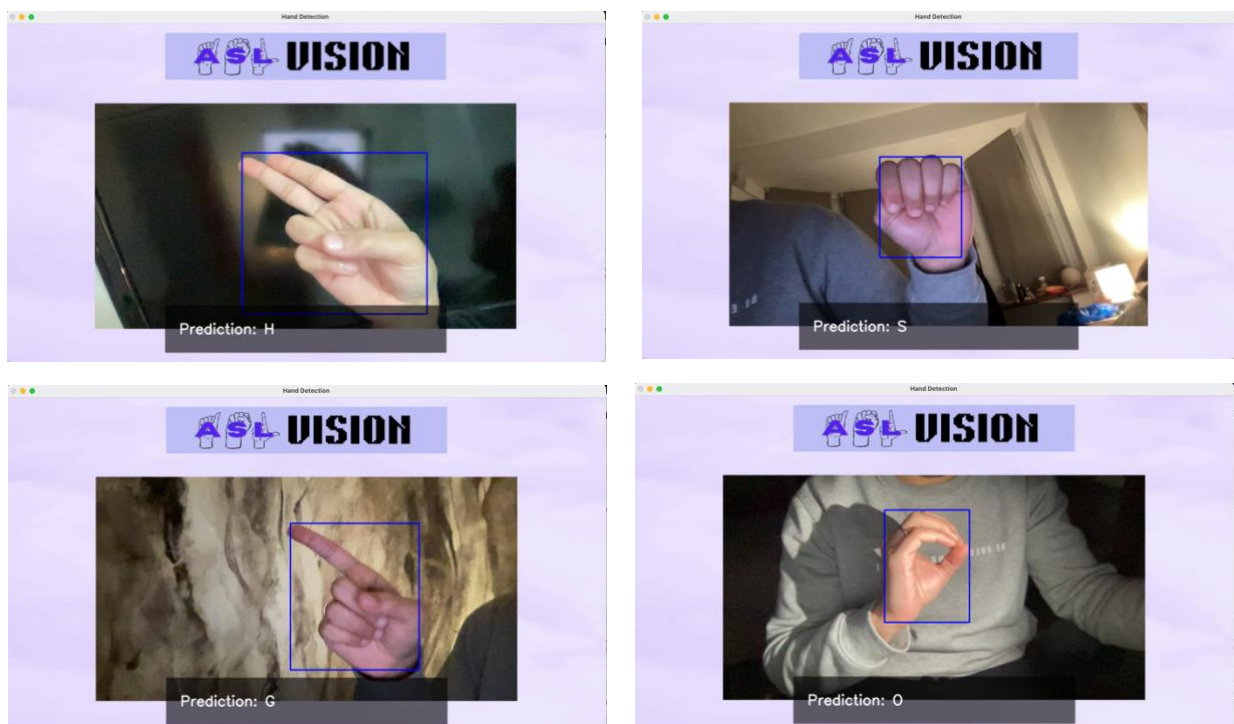
Real-Time Testing: In real-time testing, Model 5 delivered outstanding performance in varying lightning conditions and backgrounds, correctly predicting all classes with high accuracy except, the model occasionally confused the letters U and R. While it successfully classified these letters most of the time, the occasional misclassifications suggest slight limitations in distinguishing between certain letters under real-world conditions.

This exemplary performance in both the test set and real-world scenarios underscores the strength of Model 5 in recognizing sign language gestures.

4.6 User Interface for Real-Time Testing

In addition to the models' performance, a user interface (UI) was developed to facilitate real-time testing of the sign language recognition system. The UI allows users to interact with the system through a live webcam feed, enabling them to perform sign language gestures while the model classifies the gestures in real-time.

Screenshots below display the UI correctly predicting different letters in real time using the final Model 5, under various lighting conditions and backgrounds:



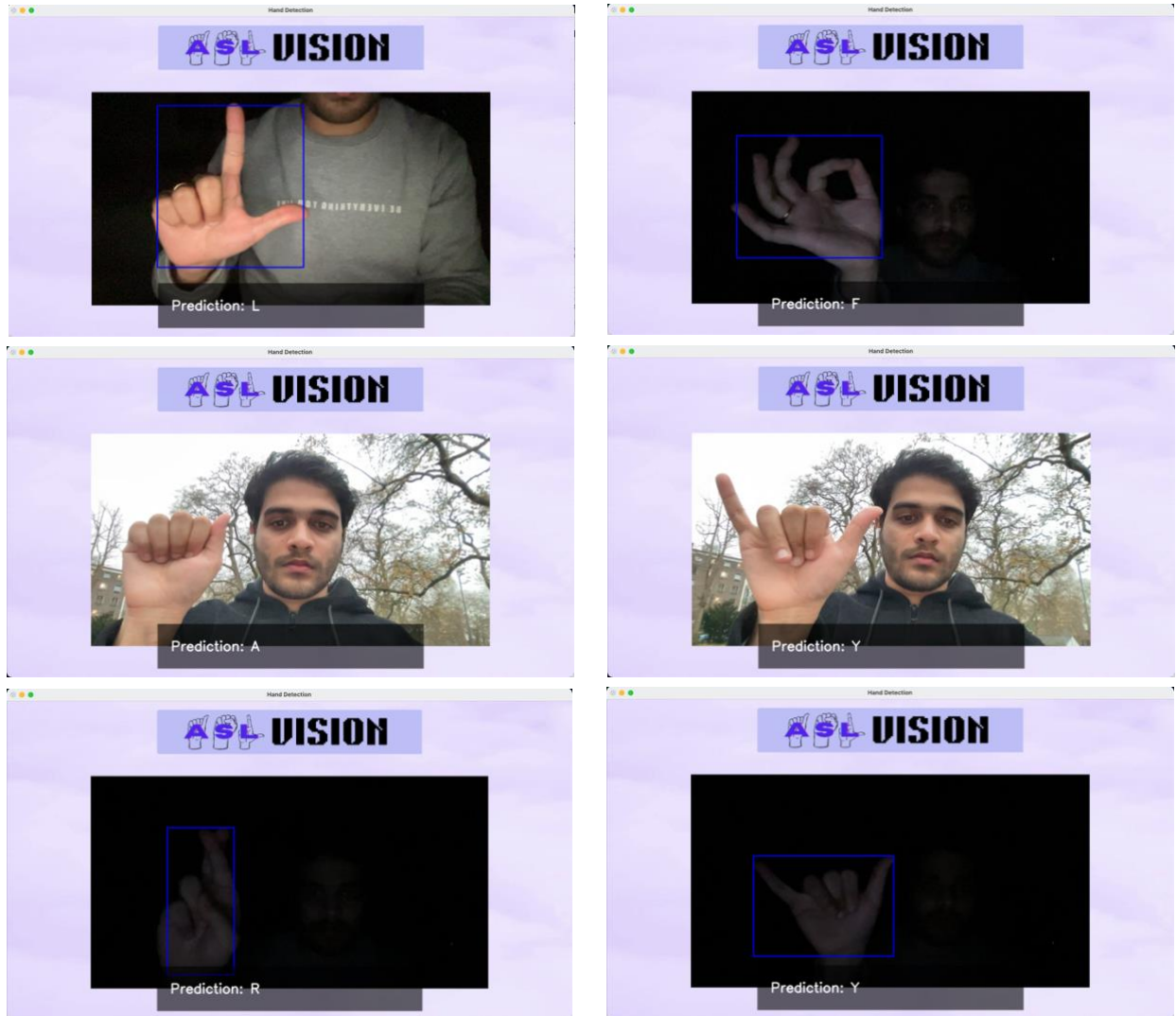


Figure 14: Screenshots of Real-time testing UI

The UI meets the specifications outlined in Section 3.6, which include the following features:

- **Real-time Gesture Recognition:** The UI displays the webcam feed, with real-time predictions of the classified gesture shown on the screen.
- **Clear Feedback:** The predicted letter is displayed prominently on the screen, along with any potential misclassification or confidence level of the prediction.
- **User-Friendly Interface:** The UI is designed to be intuitive, allowing users to quickly start testing the system without requiring complex setup or interaction.

The UI significantly contributes to the real-time testing and validation of the sign language recognition system, providing an accessible and interactive environment for users to test the models in real-world conditions.

5. Discussion

The aim of this project was to develop a robust system for recognizing American Sign Language (ASL) alphabet gestures using machine learning and computer vision techniques. Our primary objective was to improve the performance of sign language recognition systems in real-time scenarios compared to models discussed in existing research. To achieve this, four distinct approaches were chosen, each leveraging different feature extraction methods and classification models. To achieve this, four distinct approaches were designed, incorporating best practices in the field and drawing inspiration from the literature reviewed in Chapter 2. The goal was to identify the best-performing approach while addressing challenges faced by earlier works.

5.1 Model 1

For the baseline model (Model 1), which utilized the simplest feature extraction method (HOG), we observed high performance on the test set, achieving an accuracy of 99%. However, its performance in real-time testing was egregiously poor. This significant discrepancy between test set accuracy and real-time performance highlights critical challenges in model generalization to real-world data. Several factors likely contributed to this issue:

5.1.1 Factors Contributing to Poor Real-Time Performance

- a) Differences in Data Distribution (Train/Test vs. Real-Time Data)
 - **Lighting Conditions:** The training and test datasets likely featured consistent lighting, whereas real-time webcam inputs were subject to variations in brightness, shadows, and reflections.
 - **Background Variations:** The training images often had uniform or simple backgrounds, whereas real-time inputs included cluttered and diverse environments.
 - **Resolution Differences:** Webcam images might have had different resolutions or noise levels, introducing additional challenges for the model.
- b) **Overfitting to the Training/Test Set:** The extremely high accuracy on the test set suggests potential overfitting, where the model learned specific patterns in the dataset rather than generalizable features. Although stratified sampling was used for dataset splitting, this does not guarantee robust performance on entirely new types of data, such as real-time webcam inputs.

- c) **Insufficient Representation in Training Data:** The training dataset may not have sufficiently captured the variability of real-world conditions, including diverse hand shapes, orientations, sizes, and other features.

5.1.2 Factors Influencing Better Performance for Certain Letters

Interestingly, in real-time testing, certain letters—namely C, G, H, L, and N—performed slightly better than the others. This variation in performance can be attributed to both the nature of the features extracted by HOG and the inherent ambiguity of some ASL alphabet gestures:

- a) **Ambiguity in Other Letters:** Many ASL alphabet gestures share similar shapes or overlapping features, making them harder to distinguish using HOG features. For example:
- The edge structures of letters such as A and S, or I and J, appear similar. Figure 15 below shows the signs for A and S.

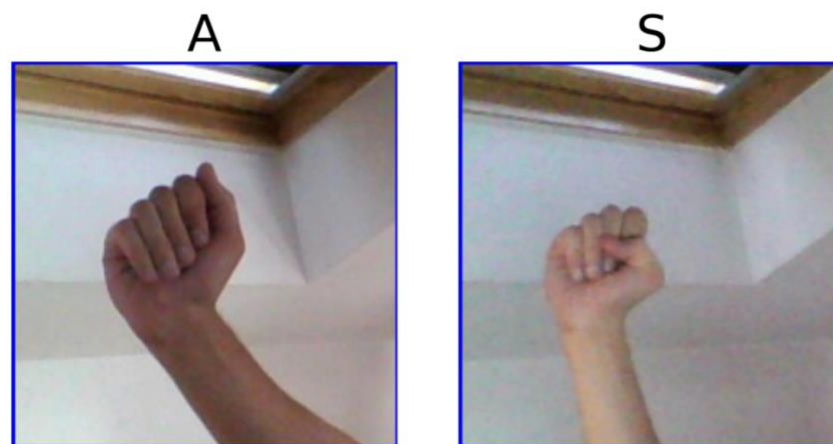


Figure 15: ASL gestures for letters A and S

- Overall shapes for letters like I, J, and Z, or D and P, can overlap, leading to confusion in predictions. Figure 16 and 17 below shows the signs for I, J, Z and P, D respectively.



Figure 16: ASL gestures for letters I, J and Z

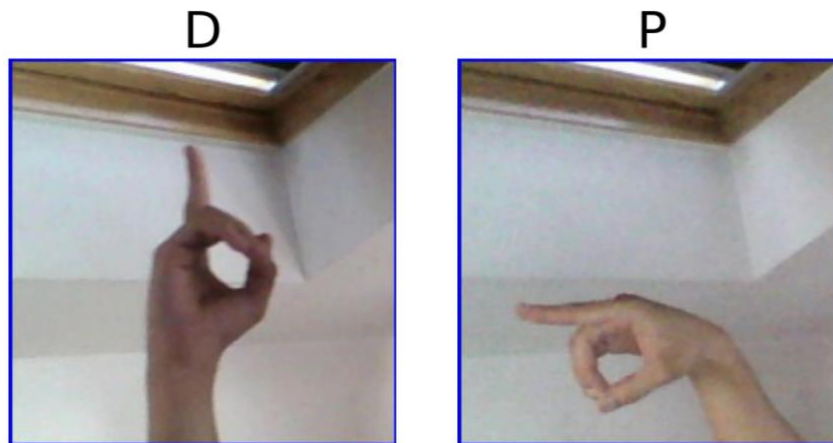


Figure 17: ASL gestures for letters I, J and Z

- b) Limitations of HOG for Complex Features: HOG primarily captures edge structures and overall shapes, which works well for alphabets with distinct visual features but struggles with:
- Complex Gestures: Letters requiring nuanced representations of finger positions.
 - Fine Details: Subtle variations in finger angles, overlaps, or small movements.
 - Overlapping Gestures: Letters with similar edge patterns, where the extracted features may not provide sufficient discriminative power.
- c) SVM Classifier's Decision Boundaries: SVM performs well when features are linearly separable in the transformed feature space. For letters like C, G, H, L, and N, the HOG features likely resulted in well-separated clusters, enabling the model to classify these gestures more effectively. For other letters, significant feature overlap likely led to poor predictions.

The results for Model 1 reveal the limitations of using traditional feature extraction methods like HOG in real-world scenarios and underscore the importance of more informative features that can capture finer details of hand gestures. This insight motivated the exploration of more sophisticated feature sets in subsequent models, as detailed in the previous sections.

5.2 Model 2

In Model 2, the feature extraction focuses exclusively on angle-based features, specifically Finger Angles and Inter-Finger Angles. These features were chosen for their ability to capture the unique geometric relationships between the hand's fingers and joints, which are essential for distinguishing different sign language gestures. This approach represents an improvement over Model 1, which used

more general features that did not emphasize the specific angular configurations of the hand.

Performance results on both the test set and during real-time testing scenarios showed that these angle-based features are highly effective for hand gesture recognition. The model demonstrated high accuracy during real-time testing, maintaining performance even in dynamic conditions.

This improvement in real-time testing accuracy stems from the more precise feature extraction method in Model 2, which captures key aspects of hand geometry, particularly the angular relationships between fingers, which are critical for sign language classification.

5.2.1 Data Loss in Feature Extraction

One of the initial observations was a significant reduction in dataset size after the Mediapipe-based feature extraction process.

The primary reasons for data loss include:

- **Nothing Class:** The "Nothing" class images, which do not contain hands, were entirely lost as Mediapipe could not detect landmarks for these samples.
- **Lighting Issues:** Dark images, shadows, or overexposed regions likely hindered Mediapipe's ability to identify hands accurately.
- **Background Clutter:** Images with complex or cluttered backgrounds may have confused the Mediapipe hand detection model.
- **Partial hand visibility:** Poorly cropped or partially occluded hands in some images.

Although the dataset was not severely skewed overall, one class was found to have significantly fewer samples, indicating a moderate imbalance. While this imbalance did not critically affect model training, it reflects the challenges of relying on automated feature extraction tools for datasets with diverse visual characteristics.

5.2.2 Factors Influencing the Better Real-Time Testing Performance Compared to Model 1

- **Robustness of Mediapipe and Angle Features:** Mediapipe hand landmarks provide pose-invariant keypoints, which are robust. Unlike HOG features in Model 1, which are highly sensitive to image quality and

environmental conditions, Mediapipe's keypoints offer more reliable data in real-time scenarios.

- **Angle-Based Features:** Finger Angles and Inter-Finger Angles are based on the relative positions of landmarks, making them resilient to changes in scale, rotation, and translation. These features are particularly effective in real-time testing, where environmental conditions can vary.
- **Simpler Feature Space:** Model 2 uses 13 angle-based features, whereas Model 1 uses the higher-dimensional HOG features. A lower-dimensional feature space reduces the risk of overfitting, improving generalization and making the model more suitable for real-time data.
- **Alignment with Real-Time Data:** The feature extraction process in Model 2 focuses on hand-related features, ignoring background noise. This approach improves the model's performance in dynamic, real-world settings where backgrounds and lighting can vary. In contrast, HOG features in Model 1 are sensitive to these variations, which can lead to performance degradation in real-time testing.
- **Reduction of Irrelevant Data:** Mediapipe automatically discards images where hands are not detected, ensuring that only relevant samples are included in training and testing. This leads to cleaner, more accurate data, which is particularly beneficial in real-time applications. Model 1, relying on HOG, does not have the same automatic filtering process, which could impact accuracy in dynamic settings.

Although Model 2 performed well, there were some limitations observed during real-time testing. The model failed to predict the gestures for the letters E, I, K, R, T, U, X, and Z correctly, suggesting that further improvements are necessary to handle these specific gestures more effectively. These challenges are addressed in the next models, where additional feature extraction techniques are incorporated to further enhance performance.

5.3 Model 3

In Model 3, we aimed to further improve the performance of real-time testing compared to Model 2 by incorporating distance-based features alongside the previously used angle-based features. The addition of these distance-based features allows the model to capture not only the relative orientations of the fingers but also their spatial relationships, enhancing its ability to distinguish gestures that may have subtle angular or positional differences. These features were compact and interpretable, enabling the model to achieve excellent

performance on the test set with an accuracy of 95.18%. However, the project also uncovered important observations during feature extraction and subsequent testing.

During feature extraction, data loss was observed, similar to the issues described in Section 5.2.1.

Despite achieving a high test set accuracy, the model's performance during real-time testing dropped drastically compared to Model 2. This suggests that the addition of distance-based features did not improve real-time performance and, in fact, may have had a detrimental effect. The distance features turned out to be misleading, possibly due to their sensitivity to variations in hand positioning or environmental factors such as background and lighting.

5.3.1 Factors Contributing to Poor Real-Time Performance

Similar to Model 1, the discrepancy between test and real-time performance can be attributed to several factors outlined in Section 5.1.1. In addition to those reasons, the following factors specific to Model 3 played a significant role:

- **Mediapipe Limitations:** While Mediapipe is a robust tool for hand landmark detection, inaccuracies in landmark detection during real-time testing can often result in incorrectly identified positions. These errors directly affected the calculated distances and angles, introducing noise into the extracted features.
- **Limited Detail in Features:** Although the distance and angle features are compact and interpretable, they may not capture the intricate details needed to differentiate between certain gestures in complex real-world scenarios, particularly for letters with similar hand configurations.
- **Noise in Real-Time Features:** Small inaccuracies in the detected hand landmarks during real-time testing propagated through the feature extraction process. These inaccuracies affected angle calculations and distance measures, which in turn compromised the SVM classifier's ability to make accurate predictions.

5.3.2 Factors Influencing Correct Prediction of Certain Letters

Interestingly, in real-time testing, the model showed relatively better performance for the letters S, T, and Y, while struggling to accurately classify most of the other letters. This trend can be explained by the unique

configurations of these gestures, which align well with the distance and angle features:

- S: The closed fist associated with the gesture for S results in compact distances between landmarks and distinct inter-finger angles, making it easier to identify.
- T: The gesture for T features a clear intersection of the thumb and index finger, which creates distinctive angular features that are easily captured by the model.
- Y: The "hang loose" gesture for Y, with the thumb and pinky finger extended, generates extreme distances between the fingertips and the wrist. These prominent features provide the model with a strong basis for accurate classification.

Despite these few instances of better performance, the overall results indicate that the model still faces challenges in capturing sufficient variability and robustness for real-time gesture recognition. These findings highlight the need for more sophisticated feature extraction techniques, which were explored in subsequent models.

5.4 Model 4

Model 4 aimed to improve upon Model 3 by refining the feature set for better real-time performance. Since distance-based features showed a detrimental effect on the model's real-time performance, they were discarded in this iteration. Instead, we focused on enhancing the angle-based features by retaining the two angle-based features from Model 2 and incorporating finger curl as a new feature. This addition was intended to address the limitations observed in Model 3, particularly in distinguishing subtle hand configurations that could be critical for accurate gesture recognition.

During feature extraction, data loss was observed, similar to the issues described in Section 5.2.1.

Model 4 achieved a test set accuracy of 91.17% and demonstrated a drastic improvement in real-time performance compared to the previous models. This improvement aligned with the intended goal of enhancing robustness in real-world conditions. The model only struggled with the letters I, R, and U, with a particular observation that I was being misclassified as J.

5.4.1 Improved Real-Time Robustness with Finger Curl

The addition of finger curl as a feature contributed to the model's improved robustness in real-time testing. This improvement can be attributed to the following factors:

- **Finger Curl Captures Pose Robustly:** Finger curl ratios are scale-invariant, meaning they are unaffected by variations in hand size, orientation, or distance from the camera. Unlike distances and angles, finger curl captures the relative pose of each finger, making it more consistent under diverse real-time conditions.
- **Better Differentiation of Similar Gestures:** Finger curl provides additional discriminatory power for gestures that have similar inter-finger angles but differ in finger configurations. Many sign language letters involve subtle variations in finger positioning that angles and distances alone cannot effectively capture. Finger curl helps the model better distinguish these gestures.
- **Noise Mitigation:** Finger curl features are less sensitive to inaccuracies in landmark detection. Even if there is minor misalignment in the detected positions of the fingertip or MCP joint, the overall curl ratio (distance from the base to the tip vs. base to PIP) remains stable. This resilience reduces the impact of noisy landmarks, which are common in real-time scenarios.
- **Complementary Features:** Finger curl complements angle-based features by providing additional information to resolve ambiguities. This synergy enables the model to correct errors caused by noisy angle computations, leading to more accurate predictions.

5.4.2 Struggling Letters in Real-Time: I, R, and U

Despite the improved real-time performance, Model 4 encountered challenges in accurately predicting I, R, and U gestures. These difficulties may stem from the following factors:

- **I and U:** Both involve extended fingers with compact overall configurations, resulting in similar finger curl features. This similarity makes them harder to differentiate.
- **R:** The "crossed fingers" pose relies heavily on inter-finger angles, which are particularly sensitive to slight hand rotations or inaccuracies in landmark detection.

- These gestures likely rely more on inter-finger angles, which remain prone to noise in real-time conditions. Their relatively less distinctive curl-based features further contribute to the model’s difficulty in identifying them accurately.

Model 4 showed that incorporating finger curl as an additional feature can greatly enhance the real-time performance of sign language recognition systems. Despite some challenges with specific gestures like I, R, and U, the model achieved the intended goal of improved robustness for real-time testing. This highlights the importance of refining the feature set to further enhance recognition accuracy.

5.5 Model 5

In Model 5, we aimed to improve real-time testing performance over Model 4 by incorporating palm orientation as an additional feature. Our primary goal was to improve predictions for the alphabets which Model 4 struggled with.

As in previous models, data loss during feature extraction was observed, with the reasons discussed in Section 5.2.1.

Model 5 achieved 94.88% accuracy on the test set, an improvement compared to Model 4. Additionally, real-time performance also showed significant improvement, with the model correctly classifying all alphabets. It occasionally struggled with R and U, but this was a clear improvement from Model 4, which misclassified I, R, and U.

The improved performance in Model 5 compared to Model 4 can be attributed to several factors:

5.5.1 Palm Orientation Captures Global Hand Pose

In Model 4, the model relied primarily on finger angles and curls. While these features were effective for many gestures, they focused on local finger positions and sometimes struggled with gestures that shared similar finger configurations (like I and J). Adding palm orientation provides information about the global hand pose, making the model more robust to small variations in finger positions and noisy landmark detections. This was particularly useful for differentiating gestures like I and J, which share a key similarity in the pinky but differ in the overall hand tilt and palm alignment.

5.5.2 The Gestures I and J

The gestures I and J share a key similarity: both involve the pinky finger. However, they differ significantly in the overall tilt of the hand and palm alignment:

- I: The pinky is vertical, and the palm's orientation aligns more with the vertical axis.
- J: The pinky is slanted, and the palm's orientation shifts accordingly. The addition of palm orientation allowed the model to better distinguish between these similar gestures, contributing to improved real-time predictions.

5.5.3 Enhanced Generalization to Real-Time Testing

Palm orientation is particularly effective in real-time scenarios because it captures 3D spatial relationships using relative vectors (e.g., between the wrist, middle finger MCP, and pinky MCP). These vectors are less sensitive to variations in lighting, hand size, or camera angle compared to 2D features like angles or distances. As a result, palm orientation helps the model become more resilient to real-world variability, improving its accuracy and robustness during real-time testing.

5.6 Conclusion and Insights from the Discussion

In summary, the results of this project have demonstrated significant progress, starting from a poorly performing model with suboptimal features to a highly accurate model with well-engineered features. This progression highlights the improvements in accuracy, responsiveness, and robustness, especially in varying real-world conditions. The objectives of the project were successfully met, as evidenced by the model's high accuracy in predicting sign language gestures across diverse backgrounds and lighting conditions. This indicates that the system can adapt well to real-time environments, ensuring smooth and efficient user experiences.

The real-time testing system has proven to be highly responsive, making fast and reliable predictions that meet the needs of users, including both accuracy and speed. The system's ability to recognize gestures in real-time with minimal delay makes it suitable for practical applications, such as communication assistance for individuals with hearing impairments.

Furthermore, the final model showed robustness in handling different environmental factors, such as varying backgrounds and lighting, ensuring the consistency of its predictions across different scenarios. This adaptability is crucial for real-world applications where conditions are rarely ideal.

We are confident in the validity of the results, as the system performed consistently across multiple test cases and real-time scenarios, except for occasional difficulties with recognizing the gestures for "R" and "U". The accuracy achieved meets our expectations, validating the effectiveness of the feature set and model used in this project.

The scope of the results is promising, particularly for future work in enhancing sign language recognition technologies. The generalisability of the system's performance across varying conditions suggests potential for widespread application in assistive technologies. However, further improvements could be made by incorporating additional contextual features, enhancing the diversity of the dataset, and optimizing the model for even better performance in challenging environments.

6. Evaluation, Reflections and Conclusions

This chapter provides a comprehensive evaluation of the project, reflecting on the key aspects of the work undertaken, the methods employed, and the overall objectives achieved. It also offers insights into the lessons learned throughout the project process, the implications of the findings, and suggestions for potential future work. Additionally, this section includes a reflective evaluation of the project, considering what could be done differently if the work were to be repeated.

6.1 Evaluation of the Project

The primary objective of this project was to develop and evaluate a real-time sign language recognition system that could accurately identify gestures in diverse and challenging environments. The project's success was contingent on several factors, including the choice of objectives, the methods employed, and the planning process.

Original Choice of Objectives

The objectives were clearly defined from the outset, focusing on creating a system that is efficient and reliable for real-time applications. These objectives were based on the recognition that factors such as background noise, lighting conditions, and hand pose variability are highly influential in determining the performance of sign language recognition systems. The project aimed to address these challenges by implementing feature extraction techniques such as computing finger angles, inter-finger angles, finger curls, and palm orientation, coupled with machine learning models that could adapt to varying input conditions.

Literature Reviewed

The literature review provided valuable insights into existing sign language recognition systems. Reviewing the work of Shin et al. (2021), who achieved notable accuracy with angle-based features, was instrumental in guiding the choice of features for the models in this project. This research highlighted the potential of angle-based features but also pointed to areas of improvement, which our approach addressed by incorporating different angle based features and additional features like finger curls and palm orientation. By combining these features and leveraging machine learning techniques, this project has expanded upon the existing body of knowledge.

Methods Used

The methods chosen for the project were appropriate for the problem at hand. The use of the ASL Alphabet Dataset, coupled with feature extraction techniques such as MediaPipe, provided a solid foundation for the recognition system. The model training pipeline was well-structured, involving careful data preprocessing, augmentation, and feature extraction, followed by model training using a range of classifiers. This methodology was validated through rigorous real-time testing, and the results indicated strong performance, validating the chosen methods.

While the deep learning models, such as CNNs, were initially considered in the project proposal, they were ultimately not used. Upon evaluating various methodologies, I realized that using CNNs would be computationally expensive and could reduce the system's computational efficiency, particularly for real-time applications. Instead, I focused on feature engineering and leveraged SVMs, which proved to be a highly effective approach, delivering excellent performance without compromising efficiency.

Planning and Execution

The project was planned methodically, with milestones set for data preprocessing, model training, and testing. The iterative approach allowed for adjustments based on intermediate results, ensuring continuous improvement. Real-time testing played a crucial role in evaluating the system's practical applicability and helped identify areas that required further optimization. The structured approach ensured that the project was completed within the specified time frame, and all objectives were met.

6.2 General Conclusions and Contribution

This project successfully developed a sign language recognition system that is capable of real-time predictions in a variety of environmental conditions. The model demonstrated high accuracy and robustness, successfully addressing the challenges of real-time applications, including variable backgrounds, lighting, and hand poses. A key contribution of this project is the engineering of several novel features, which significantly contributed to the success of the final model. Additionally, the system's ability to recognize sign language gestures with minimal delay makes it a viable tool for communication assistance.

The system's performance met the objectives, particularly in real-world testing scenarios, where the accuracy and responsiveness of the model were crucial. The combination of finger angles, inter-finger angles, finger curls, and palm orientation as features proved effective in capturing the necessary information for accurate gesture classification. Model 5, which incorporated these features, was the most successful, achieving high levels of performance in real-time applications.

6.3 Implications and Further Work

The success of this project opens up several potential avenues for further development in the field of sign language recognition. One significant area for improvement is the testing process, which could be enhanced by testing the system with a wider range of users. This would help confirm the system's adaptability to different hand shapes, gestures, and individual variations, ensuring it is more universally applicable.

Another key improvement would be to train the model with a larger and more diverse dataset, ideally combining multiple datasets. This would improve the model's ability to generalize across various real-world conditions and hand variations, further enhancing its robustness. Expanding the dataset to include a broader range of sign language gestures beyond the alphabet would also make the system more capable of recognizing complex sign language expressions, addressing the need for more comprehensive communication assistance.

Additional research could explore the integration of the sign language recognition system into assistive devices or mobile applications. By embedding this technology into user-friendly platforms, such as smartphones or wearable devices, the system could provide real-time communication support for individuals with hearing impairments, increasing accessibility and promoting independence.

These future directions will help improve the system's versatility, scalability, and applicability, ensuring it can serve as a valuable tool in real-world settings.

6.4 Reflective Evaluation

What Was Learned from the Project Process?

Throughout the course of the project, valuable lessons were learned regarding the complexity of real-time machine learning systems. One key takeaway was the importance of dataset quality and diversity. A more varied dataset,

representing a wider range of hand poses, gestures, and environmental factors, would likely improve the system's generalizability. Another lesson was the significance of feature engineering in improving model performance. The exploration of multiple feature extraction methods, including HOG and MediaPipe, proved critical in achieving high accuracy.

What Would I Do Differently?

If I were to start this project again, I would experiment with Convolutional Neural Networks (CNNs) to explore their potential in improving classification accuracy for sign language recognition. CNNs are known for their ability to automatically extract spatial features, and incorporating them could enhance the model's capability to handle more complex variations in gestures.

Another improvement would be increasing the variety and size of the dataset, including a combination of multiple datasets, to ensure broader coverage of possible real-world scenarios, especially for gesture variations and environmental changes. This would help the model generalize better and improve its robustness in diverse conditions.

Additionally, incorporating feedback from potential users, such as individuals with hearing impairments, could provide valuable insights into the system's practical usability and areas for improvement. This user-centric approach would help align the system with the needs of its intended audience, making it more effective and user-friendly.

6.5 Final Conclusion

In conclusion, this project has successfully achieved its goal of developing a real-time sign language recognition system that is both accurate and efficient. The methods employed were well-suited to the problem, and the results demonstrate the potential for real-world application of the system. The findings highlight the importance of feature engineering, model selection, and real-time testing in creating effective sign language recognition systems. Although the system occasionally experiences minor difficulties in recognizing the gestures for "R" and "U," the overall performance outshines these challenges, demonstrating the system's reliability. Moving forward, there is significant potential to enhance the system's capabilities, making it even more adaptable and robust for a wider range of users and environments.

Glossary

Word	Definition
MCP (Metacarpophalangeal Joint)	The joint at the base of each finger where the finger connects to the hand. It allows the fingers to bend and straighten, as well as move side to side.
PIP (Proximal Interphalangeal Joint)	The middle joint of each finger that lets the finger bend and straighten. It's located between the knuckle and the tip of the finger.
Tip	The very end of a finger, also referred to as the fingertip. It's the outermost part of the finger used for touching or interacting with objects.
Stratified Sampling	A sampling technique where the data is divided into distinct strata or groups, ensuring that each group is proportionally represented in the sample. This method is commonly used to maintain class balance in datasets.
Data Augmentation	A technique used in machine learning to artificially expand a dataset by applying transformations such as rotation, flipping, cropping, or colour adjustment to the original data, improving model robustness and generalization.
Overfitting	A phenomenon in machine learning where a model performs exceptionally well on training data but fails to generalize to unseen data due to excessive complexity or memorization of noise in the training set.
Generalization	The ability of a machine learning model to perform well on unseen data by accurately capturing underlying patterns in the training data without overfitting to noise or specific instances.

Computer Vision	A field of artificial intelligence that enables machines to interpret and understand visual information from the world, such as images or videos, by mimicking human vision capabilities.
Tkinter	A Python library used for creating graphical user interfaces (GUIs). It provides tools to design and build interactive windows, buttons, text boxes, and other elements for user interaction with software applications.
Hand Landmarks	Key points or specific locations on the hand that are used to define the position and movement of the hand and fingers. These landmarks typically include joints such as the wrist, knuckles, and fingertips, and are identified using computer vision techniques for hand gesture recognition and analysis.
Frame Rate	The number of frames (images) displayed or processed per second in a video or real-time system. In the context of sign language recognition, a higher frame rate indicates smoother and more continuous tracking of hand movements, which is essential for real-time predictions. It is usually measured in frames per second (FPS).
Latency	The time delay between an input being given (e.g., a hand gesture) and the system's response (e.g., recognizing the gesture). In real-time systems, low latency is crucial for providing immediate feedback. High latency can cause delays in predictions, leading to poor user experience.

References

- [1] Shin J, Matsuoka A, Hasan MAM, Srizon AY. American Sign Language Alphabet Recognition by Extracting Feature from Hand Pose Estimation. *Sensors (Basel)*. 2021 Aug 31;21(17):5856. doi: 10.3390/s21175856.
- [2] Kołodziej, Marcin & Szypuła, Ernest & Majkowski, Andrzej & Rak, Remigiusz. (2022). Using deep learning to recognize the sign alphabet. *Przegląd Elektrotechniczny*. 06/2022. 32. 10.15199/48.2022.06.06.
- [3] Verma, Aditya & Singh, Gagandeep & Meghwal, Karnim & Ramji, Banawath & Dadheech, Praveen. (2024). Enhancing Sign Language Detection through Mediapipe and Convolutional Neural Networks (CNN). 10.48550/arXiv.2406.03729.
- [4] Adeyanju, Ibrahim & Bello, Oluwaseyi & Azeez, Adekunle. (2022). Development of an American Sign Language Recognition System using Canny Edge and Histogram of Oriented Gradient. *Nigerian Journal of Technological Development*. 19. 195-205. 10.4314/njtd.v19i3.2.
- [5] Marais, Marc & Brown, Dane & Connan, James & Bobby, Alden. (2022). An Evaluation of Hand-Based Algorithms for Sign Language Recognition. 1-6. 10.1109/icABCD54961.2022.9856310.
- [6] Akash. ASL Alphabet. Kaggle. <https://www.kaggle.com/datasets/grassknoted/asl-alphabet/data>
- [7] Suharjito, Suharjito & Wiryana, Fanny & Zahra, Amalia. (2018). Feature Extraction Methods in Sign Language Recognition System: A Literature Review. 11-15. 10.1109/INAPR.2018.8626857
- [8] J. Debnath and P. J. I R, "Real-Time Gesture Based Sign Language Recognition System," 2024 International Conference on Advances in Data Engineering and Intelligent Computing Systems (ADICS), Chennai, India, 2024, pp. 01-06, doi: 10.1109/ADICS58448.2024.10533518.
- [9] Sarah Alyami, Hamzah Luqman, Mohammad Hammoudeh. Reviewing 25 years of continuous sign language recognition research: Advances, challenges, and prospects (2024). <https://doi.org/10.1016/j.ipm.2024.103774>.
- [10] Yulius Obi, Kent Samuel Claudio, Vetri Marvel Budiman, Said Achmad, Aditya Kurniawan. Sign language recognition system for communicating to people with disabilities (2023). <https://doi.org/10.1016/j.procs.2022.12.106>.
- [11] Cooper, Helen & Holt, Brian & Bowden, Richard. (2011). Sign Language Recognition. 10.1007/978-0-85729-997-0_27.
- [12] Google AI, 2024. MediaPipe Hand Landmarker. Available at: https://ai.google.dev/edge/mediapipe/solutions/vision/hand_landmarker [Accessed 15 Nov. 2024]
- [13] Google AI, 2024. Hand landmarks detection guide for Python. Available at: https://ai.google.dev/edge/mediapipe/solutions/vision/hand_landmarker/python [Accessed 27 Oct. 2024].

Appendix A: Project Proposal

Sign Language Recognition using Deep Learning and Computer Vision

Baibhav Datta, MSc Data Science, City University of London

Introduction

Sign language serves as a vital means of communication for individuals with hearing impairments, enabling them to express themselves, interact with others, and access information. However, despite its significance, sign language interpretation often relies on human interpreters, leading to limitations such as availability, cost, and accuracy. This underscores the need for automated sign language recognition systems to bridge this communication gap effectively.

By enabling the translation of sign language alphabets into text or speech, a sign language recognition system bridges the communication gap between individuals who use sign language and those who do not. It enhances social inclusion, facilitates education, and improves access to essential services such as healthcare and emergency response. Moreover, sign language recognition empowers individuals with hearing impairments to express themselves more fluently and participate fully in various aspects of daily life, thereby promoting equality and diversity in society.

This project aims to develop a robust system for recognising sign language alphabets and forming words based on the signs captured through a webcam. By leveraging computer vision and deep learning techniques, this project seeks to bridge this gap by accurately interpreting sign language gestures captured in real-time. The focus will primarily be on recognising individual sign language alphabets and seamlessly combining them to form coherent words, facilitating more inclusive communication channels for individuals with hearing impairments.

To achieve this, we will utilise the ASL Alphabet dataset available on Kaggle, which comprises a comprehensive collection of images representing each letter of the ASL alphabet. By training models on this dataset, we aim to bridge communication barriers for individuals with hearing impairments, providing them with a tool to interact more effectively in various settings. The dataset comprises 87,000 images, each sized 200x200 pixels, distributed across 29 classes. Among these classes, 26 represent the letters A-Z, while the remaining 3 classes are designated for SPACE, DELETE, and NOTHING. The inclusion of these auxiliary classes enhances the dataset's utility in real-time applications and classification tasks.

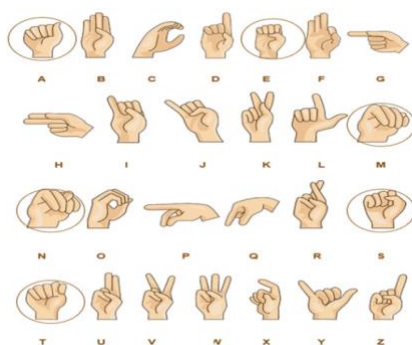


Fig 1: Illustration of signs for alphabets

The project will involve data preprocessing, model training, interface design, and testing. The system will focus on recognising ASL gestures corresponding to individual alphabets. Additionally, the system will aim to achieve real-time performance to ensure practical usability.

Critical Context

The proposed project addresses the critical need for improved communication accessibility for impaired hearing individuals. By recognising and interpreting ASL gestures, the system will facilitate communication between impaired hearing individuals and those who may not understand sign language. This technology has the potential to enhance inclusivity, foster understanding, and promote equal opportunities for impaired hearing individuals in various settings, including educational institutions, workplaces, and public spaces.

The key products of this work will include a trained machine learning model capable of recognising ASL alphabet gestures, along with a user-friendly interface for real-time interpretation. This project will benefit individuals with hearing impairments by facilitating smoother communication and improving their access to information and services.

Previous work on sign language recognition has seen advancements in various approaches and techniques. Researchers have explored both traditional machine learning methods and deep learning architectures for feature extraction and classification tasks. Some studies have focused on leveraging computer vision techniques like Histogram of Oriented Gradients (HOG) and Convolutional Neural Networks (CNNs) for recognising hand gestures. Additionally, research has highlighted the importance of large-scale datasets and annotated corpora for training deep learning models effectively. Transfer learning techniques have also been explored to adapt pre-trained models to sign language recognition tasks, improving accuracy and performance.

To establish the foundation for this endeavour, it is essential to delve into the existing research landscape surrounding sign language recognition, machine learning, and computer vision. Here are some insights derived from relevant scientific literature:

1) Alsharif B et al. (2023), "Deep Learning Technology to Recognise American Sign Language Alphabet":

- This study has highlighted the importance of real-time processing and interactive interfaces in sign language recognition. By integrating deep learning and image processing techniques, these studies have paved the way for practical applications in assistive technology.
- The authors experimented with several well-known pre-trained models, including AlexNet, ConvNeXt, EfficientNet, and ResNet. The study's key findings indicate that the ResNet-50 model outperformed the other models.
- Building upon these findings, the proposed project aims to develop a robust sign language recognition system to enhance communication accessibility for individuals with hearing impairments.

2) I.A. Adeyanju et al. (2021), "Machine learning methods for sign language recognition: A critical review and analysis":

- This paper provides valuable insights into the field of sign language recognition. The authors conduct a comprehensive review of machine learning techniques applied to this domain, emphasising the importance of accurate recognition for facilitating communication and inclusion among the hearing-impaired community.

- The review critically evaluates various approaches, including traditional machine learning algorithms and deep learning architectures, highlighting their strengths and limitations. Additionally, the paper discusses the challenges associated with dataset collection, model training, and real-time implementation.
- Overall, this review serves as a foundational resource for understanding the current landscape of sign language recognition and guiding future research endeavours in this area.

3) Kankariya et al. (2024), "Sign Language Gestures Recognition using CNN and Inception v3":

- The study focuses on sign language gesture recognition using convolutional neural networks (CNNs) and the Inception v3 architecture.
- The research explores the effectiveness of deep learning techniques in recognising sign language gestures, leveraging the capabilities of CNNs and the feature extraction power of the Inception v3 model.
- By training and evaluating these models on sign language datasets, the paper provides insights into the performance and accuracy of CNN-based approaches for gesture recognition tasks.
- Additionally, it discusses the advantages of using sophisticated architectures like Inception v3 for extracting discriminative features from sign language images, thereby contributing to the advancement of sign language recognition technology.

4) Ahmed et al. (2022), "DeepASLR: A CNN based human computer interface for American Sign Language recognition for hearing-impaired individuals":

- This paper proposes a CNN-based system for ASL recognition. Key findings suggest promising accuracy in classifying ASL alphabets.
- The study highlights the potential for future enhancements through training with larger datasets to improve model performance.

5) Zhang and Jiang (2024), "Recent Advances on Deep Learning for Sign Language Recognition":

- Zhang and Jiang provide insights into recent advances in deep learning for SLR.
- Their study explores the application of state-of-the-art deep learning models and techniques, such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs), for sign language recognition tasks.
- They emphasise the importance of large-scale datasets and annotated corpora for training deep learning models effectively.

These insights provide a comprehensive understanding of the challenges and opportunities in sign language recognition. By synthesising these findings, we can inform the proposed approach and guide the selection of appropriate methodologies and techniques to address the identified research gaps.

Approaches

The core approach involves a comprehensive comparison of various pre-trained models that have been utilized in different research studies for sign language recognition. By systematically

evaluating and presenting a comparative analysis of these models, the objective is to identify the best-performing model in terms of accuracy, computational efficiency, and overall reliability. Through this comparative analysis, the project aims to highlight the strengths and weaknesses of each model, ultimately determining the most effective approach for sign language recognition.

1. Data Preprocessing:

- The ASL Alphabet dataset from Kaggle will serve as our primary data source, comprising images representing the American Sign Language alphabet.
- Preprocessing steps will include:
 - Image resizing: Standardising image dimensions to a suitable size for model input.
 - Normalization: Scaling pixel values to a range suitable for model training.
 - Augmentation: Generating additional training samples through techniques like rotation, flipping, and zooming to enhance model generalisation.

2. Feature Extraction:

- Feature extraction will focus on capturing discriminative information from the images.
- Feature extraction is performed using pre-trained CNN models like DenseNet, ResNet, and Inception. These models are adept at extracting hierarchical features from input sign language gesture images.
- By passing images through these models, we capture high-level abstract features that encode the visual characteristics of gestures. These features serve as inputs for subsequent classification layers.

3. Model Selection:

Several models will be considered for image classification, including:

Utilize Convolutional Neural Networks (CNNs):

- Leveraging architectures like ResNet, Inception and DenseNet for feature extraction and classification.

Incorporate Transfer Learning:

- By leveraging pre-trained models trained on large-scale datasets, the model can inherit knowledge from related tasks and adapt it to the specific domain of sign language recognition.
- Transfer learning accelerates the training process and improves recognition performance, particularly when annotated sign language data is limited. By fine-tuning pre-trained models, the project aims to achieve higher accuracy with reduced computational resources.

Baseline Models

- Support Vector Machines (SVMs) and Multi-Layer Perceptrons (MLPs): Employing traditional machine learning models for comparison and benchmarking.

Convolutional Neural Networks (CNNs):

CNNs are a class of deep neural networks that have revolutionised the field of computer vision. They consist of multiple layers of interconnected neurons, including convolutional layers, pooling layers, and fully connected layers. CNNs are specifically designed to handle spatially structured data, such as images, by learning hierarchical representations of features directly from the pixel values.

Transfer Learning:

Transfer learning is a machine learning technique where a model trained on one task is fine-tuned or adapted to another related task. In the context of CNNs, transfer learning involves taking a pre-trained model, such as VGG, ResNet, or Inception, which has been trained on a large dataset (e.g., ImageNet), and fine-tuning it on a smaller dataset, such as the ASL dataset. By leveraging the learned features from the pre-trained model, transfer learning allows for faster training and improved performance on the target task.

ResNet (Residual Neural Network):

ResNet, introduced by Microsoft Research, revolutionised deep learning architectures by addressing the challenge of vanishing gradients in very deep neural networks. Traditional deep networks suffer from degradation issues, where adding more layers leads to decreased accuracy due to vanishing gradients. ResNet tackles this problem by introducing skip connections or residual connections. These connections enable the direct flow of information from earlier layers to later layers, allowing the network to bypass certain layers. As a result, ResNet architectures can effectively train very deep networks with hundreds of layers, achieving state-of-the-art performance in various computer vision tasks, including image classification, object detection, and segmentation.

Inception (GoogLeNet):

Inception, developed by Google, is characterised by its innovative inception modules, which are designed to capture features at multiple scales and resolutions. These modules consist of parallel convolutional layers with different filter sizes, allowing the network to extract diverse features simultaneously. By leveraging this multi-scale feature extraction, Inception architectures achieve superior performance in tasks like image classification and object detection, where capturing both fine-grained and coarse-grained features is essential for accurate recognition. Inception models are known for their efficiency and effectiveness, making them widely used in various deep learning applications.

DenseNet:

DenseNet is a deep learning architecture renowned for its densely connected layers, which establish direct connections between all layers in a feed-forward manner. Unlike traditional CNNs, where each layer is connected only to its subsequent layers, DenseNet maximises feature reuse and gradient flow throughout the network. This dense connectivity facilitates efficient information propagation, enabling the network to learn richer representations of input data. DenseNet architectures excel in tasks like image classification and segmentation, where robust feature extraction is crucial for accurate and reliable predictions. In the context of sign language recognition, DenseNet offers efficient feature extraction and robust performance, contributing to more accurate gesture classification and improved communication accessibility for the hearing-impaired.

Support Vector Machines (SVMs) and Multi-Layer Perceptrons (MLPs):

SVMs and MLPs are traditional machine learning models commonly used for classification tasks. SVMs are particularly effective for binary classification problems, while MLPs, which consist of multiple layers of interconnected neurons, are more suitable for multi-class classification problems. In the context of sign language recognition, SVMs and MLPs can be trained on handcrafted features extracted from the images or on features learned from deep neural networks using transfer learning. These models provide a baseline for comparison with more complex deep learning architectures like CNNs.

4. Evaluation Strategy:

- The dataset will be split into training, validation and test sets using stratified sampling to ensure class balance.
- Model performance will be evaluated using standard metrics such as accuracy, precision, recall, and F1-score on the test set.
- Confusion matrices will provide insights into the model's behaviour and class-wise performance.
- Additionally, ROC curves will be plotted to assess the model's ability to discriminate between classes.

5. Ethical Considerations:

- Ethical considerations will be paramount throughout the project, ensuring the well-being and privacy of all individuals involved.
- Anonymity of data will be preserved for any individuals involved in the dataset creation process.
- Measures will be taken to address any biases inherent in the dataset or model predictions.

By employing this comprehensive approach, we aim to develop a robust sign language recognition system that not only achieves high accuracy but also upholds ethical standards and considerations. Each step will be meticulously executed to ensure the reliability and effectiveness of the final model.

Real-Time Testing:

In the final stage of the project, the developed sign language recognition model will undergo real-time testing to assess its performance under practical conditions. Real-time data will be captured through the webcam, allowing users to input sign language gestures directly into the system. The captured data will then be pre-processed and fed into the trained model for prediction. The model's prediction for the sign language gesture will be displayed on the screen in real-time, providing immediate feedback to the user. This real-time testing phase aims to evaluate the model's accuracy, speed, and robustness in recognising sign language gestures in dynamic environments. Additionally, it serves as a validation step to ensure that the developed system meets the desired objectives and usability requirements for practical use.

Meetings with Project Supervisor:

Regular meetings with the project supervisor will be scheduled to ensure effective communication, progress tracking, and alignment with project objectives. These meetings will serve as opportunities to discuss project milestones, address any challenges or concerns, and receive guidance and feedback on project tasks. Additionally, ad-hoc meetings may be scheduled as needed to address urgent issues or provide updates on significant developments. Meeting agendas and summaries will be prepared in advance to ensure that discussions remain focused and productive. The project supervisor's input and insights will be invaluable in guiding the project towards successful completion and meeting its objectives.

Work Plan

The detailed work plan mentioned below outlines the sequential steps involved in each phase of the project, from proposal writing to final submission, ensuring systematic execution and successful completion of the sign language recognition project.

1. Project Proposal

- **Literature Review:** Conduct an in-depth literature review to understand existing research, methodologies, and findings related to sign language recognition.
- **Write Project Proposal:** Draft a comprehensive project proposal detailing the objectives, scope, methodologies, and expected outcomes of the sign language recognition project.
- **Submit Project Proposal:** Submit the finalised project proposal to the relevant stakeholders or supervisors for approval and feedback.

2. Coding

- **Project Initiation:** Set up the development environment, including installing necessary libraries and frameworks, creating project directories, and configuring version control.
- **Data Preprocessing:** Clean and preprocess the dataset, including data normalization and augmentation.
- **Model Development and Training:** Implement machine learning or deep learning models for sign language recognition, train the models using the pre-processed data, and fine-tune the model hyper-parameters for optimal performance.
- **User Interface Development (Time Permitting):** If time and resources allow, a user interface will be developed for the sign language recognition system, integrating the trained model for real-time prediction. The inclusion of the UI will be contingent on the successful completion of the core data science tasks.
- **Evaluation and Testing:** Evaluate the performance of the developed models using appropriate evaluation metrics, conduct rigorous testing to ensure the reliability and accuracy of the system.

3. Real-Time Processing

- **Capturing real-time data:** Implement mechanisms to capture real-time sign language gestures using cameras or sensors.
- **Preprocessing & Feature Extraction:** Preprocess the real-time data, extract relevant features, and prepare it for input to the trained model.
- **Testing Model on real-time data:** Test the trained model with real-time data to assess its performance under real-world conditions.

4. Documentation

- **Report Writing:** The report writing process will commence as soon as the project commences, ensuring that documentation is conducted concurrently with project activities. This approach allows for real-time documentation of methodologies, findings, and progress throughout the project lifecycle. As milestones are achieved and results obtained, they will be promptly documented in the report, facilitating a comprehensive overview of the project's evolution. By starting the report writing early in the project timeline, it ensures that all aspects of the project are thoroughly documented, and any issues or challenges encountered are addressed in a timely manner. This proactive approach to report writing ensures that the final report is comprehensive, accurate, and reflective of the entire project journey.

5. Project Completion and Submission

- **Review entire code and report:** Conduct a thorough review of the codebase and project report to ensure accuracy, completeness, and adherence to project objectives.
- **Identify scopes of improvements:** Identify areas of improvement or future enhancements based on the project outcomes and feedback received.
- **Apply planned enhancements:** Implement planned enhancements or optimisations to further improve the sign language recognition system before final submission.

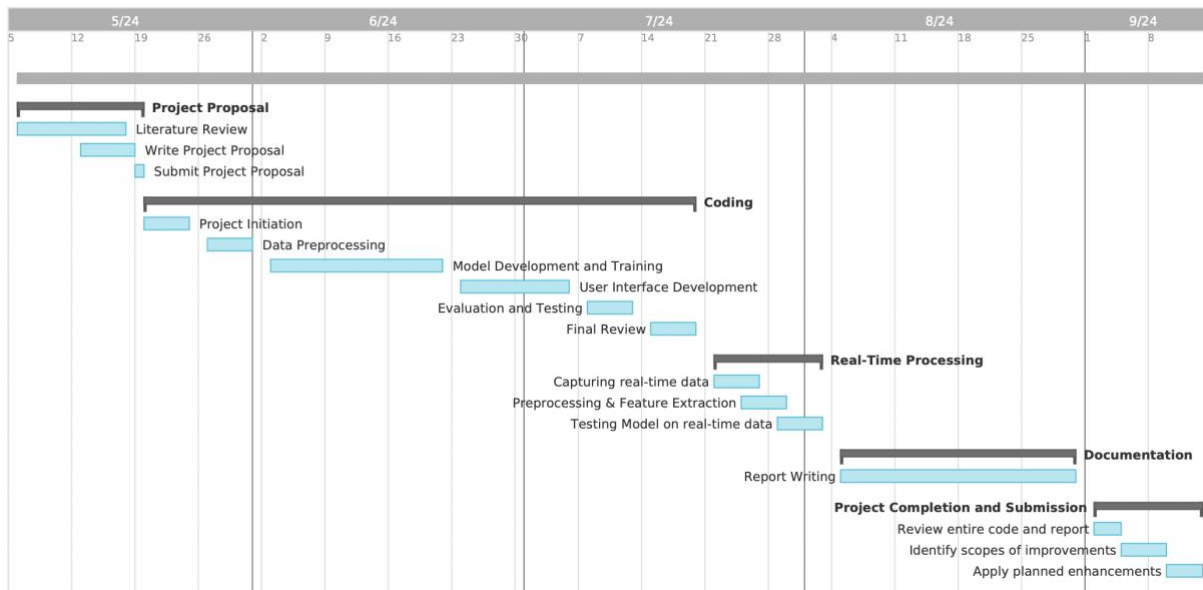


Fig 2: Project Timeline

Risks

The following risk register outlines potential risks, their likelihood, impact, and mitigation strategies, ensuring proactive measures are in place to mitigate adverse outcomes and safeguard the project's success.

Risk Description	Likelihood	Impact	Mitigation Plan	Recovery Plan
Model Overfitting	Medium	High	Regularly monitor model performance during training and validation phases. Implement early stopping and regularization techniques to prevent overfitting.	Re-evaluate model architecture and hyperparameters, conduct additional data preprocessing, and retrain the model using different optimization strategies.
Hardware Limitations	Medium	Medium	Assess hardware requirements for model training and real-time processing. Explore cloud-based solutions or hardware upgrades to address performance bottlenecks.	Optimize algorithms and codebase to minimize resource consumption.
User Interface Compatibility Issues	Low	Medium	Conduct thorough testing across different devices and platforms to ensure compatibility and responsiveness of the user interface.	Develop contingency plans to address compatibility issues promptly through software patches or updates.
Code Loss or Overwriting	Low	High	Implement version control systems (e.g., Git) and regular backups to prevent loss of code. Educate team members on best practices for code management.	Restore from the latest backup or repository version, and establish protocols to prevent future incidents.
Lengthy Model Training	Medium	High	Optimize model architecture, hyperparameters, and training strategies to reduce training time. Explore distributed computing or GPU acceleration for faster training.	Implement techniques such as transfer learning or model checkpointing to resume training from the last checkpoint in case of interruptions.

References

1. Alsharif B, Altaher AS, Altaher A, Ilyas M, Alalwany E. (2023) *Deep Learning Technology to Recognize American Sign Language Alphabet*. Available at: <https://doi.org/10.3390/s23187970>
2. Adeyanju, Ibrahim & Bello, Oluwaseyi & Adegboye, Mutiu. (2021). Machine learning methods for sign language recognition: A critical review and analysis. Available at: <http://dx.doi.org/10.1016/j.iswa.2021.200056>
3. Chavan, Shruti & Yu, Xinrui & Saniie, Jafar. (2021). *Convolutional Neural Network Hand Gesture Recognition for American Sign Language*. Available at: <http://dx.doi.org/10.1109/EIT51626.2021.9491897>
4. Ahmed KASAPBAŞI, Ahmed Eltayeb AHMED ELBUSHRA, Omar AL-HARDANEE, Arif YILMAZ (2022). *DeepASLR: A CNN based human computer interface for American Sign Language recognition for hearing-impaired individuals*. Available at: <https://doi.org/10.1016/j.cmpbup.2021.100048>
5. Zhang, Yanqiong & Jiang, Xianwei. (2024). *Recent Advances on Deep Learning for Sign Language Recognition*. Available at: <http://dx.doi.org/10.32604/cmes.2023.045731>
6. S. X. Thong, E. L. Tan and C. P. Goh (2024). *Sign Language to Text Translation with Computer Vision: Bridging the Communication Gap*. Available at: <https://doi.org/10.1109/ICDXA61007.2024.10470532>
7. K. -Y. Fok, N. Ganganath, C. -T. Cheng and C. K. Tse (2015). *A Real-Time ASL Recognition System Using Leap Motion Sensors*. Available at: <https://doi.org/10.1109/CyberC.2015.81>
8. Sulfayanti, Dewiani and Armin Lawi (2016). *A real time alphabets sign language recognition system using hands tracking*. Available at: <https://doi.org/10.1109/CyberneticsCom.2016.7892569>
9. M. Kumar, P. Gupta, R. K. Jha, A. Bhatia, K. Jha and B. K. Shah (2021). *Sign Language Alphabet Recognition Using Convolution Neural Network*. Available at: <https://doi.org/10.1109/ICICCS51141.2021.9432296>
10. P. T. Krishnan and P. Balasubramanian (2019). *Detection of Alphabets for Machine Translation of Sign Language Using Deep Neural Net*. Available at: <https://doi.org/10.1109/IconDSC.2019.8816988>

Research Ethics Review Form: BSc, MSc and MA Projects

Computer Science Research Ethics Committee (CSREC)

<http://www.city.ac.uk/department-computer-science/research-ethics>

Undergraduate and postgraduate students undertaking their final project in the Department of Computer Science are required to consider the ethics of their project work and to ensure that it complies with research ethics guidelines. In some cases, a project will need approval from an ethics committee before it can proceed. Usually, but not always, this will be because the student is involving other people ("participants") in the project.

In order to ensure that appropriate consideration is given to ethical issues, all students must complete this form and attach it to their project proposal document. There are two parts:

PART A: Ethics Checklist. All students must complete this part.

The checklist identifies whether the project requires ethical approval and, if so, where to apply for approval.

PART B: Ethics Proportionate Review Form. Students who have answered "no" to all questions in A1, A2 and A3 and "yes" to question 4 in A4 in the ethics checklist must complete this part. The project supervisor has delegated authority to provide approval in such cases that are considered to involve MINIMAL risk. The approval may be **provisional** – identifying the planned research as likely to involve MINIMAL RISK. In such cases you must additionally seek **full approval** from the supervisor as the project progresses and details are established. **Full approval** must be acquired in writing, before beginning the planned research.

A.1 If you answer YES to any of the questions in this block, you must apply to an appropriate external ethics committee for approval and log this approval as an External Application through Research Ethics Online - https://ethics.city.ac.uk/		<i>Delete as appropriate</i>
1	Does your research require approval from the National Research Ethics Service (NRES)? <i>e.g. because you are recruiting current NHS patients or staff?</i> <i>If you are unsure try - https://www.hra.nhs.uk/approvals-amendments/what-approvals-do-i-need/</i>	NO
1	Will you recruit participants who fall under the auspices of the Mental Capacity Act? <i>Such research needs to be approved by an external ethics committee such as NRES or the Social Care Research Ethics Committee - http://www.scie.org.uk/research/ethics-committee/</i>	NO
1	Will you recruit any participants who are currently under the auspices of the Criminal Justice System, for example, but not limited to, people on remand, prisoners and those on probation? <i>Such research needs to be authorised by the ethics approval system of the National Offender Management Service.</i>	NO
A.2 If you answer YES to any of the questions in this block, then unless you are applying to an external ethics committee, you must apply for approval from the Senate Research Ethics Committee (SREC) through Research Ethics Online - https://ethics.city.ac.uk/		<i>Delete as appropriate</i>
2	Does your research involve participants who are unable to give informed consent? <i>For example, but not limited to, people who may have a degree of learning disability or mental health problem, that means they are unable to make an informed decision on their own behalf.</i>	NO
2	Is there a risk that your research might lead to disclosures from participants concerning their involvement in illegal activities?	NO

2 . 3	Is there a risk that obscene and or illegal material may need to be accessed for your research study (including online content and other material)?	NO
2 . 4	Does your project involve participants disclosing information about special category or sensitive subjects? <i>For example, but not limited to: racial or ethnic origin; political opinions; religious beliefs; trade union membership; physical or mental health; sexual life; criminal offences and proceedings</i>	NO
2 . 5	Does your research involve you travelling to another country outside of the UK, where the Foreign & Commonwealth Office has issued a travel warning that affects the area in which you will study? <i>Please check the latest guidance from the FCO - http://www.fco.gov.uk/en/</i>	NO
2 . 6	Does your research involve invasive or intrusive procedures? <i>These may include, but are not limited to, electrical stimulation, heat, cold or bruising.</i>	NO
2 . 7	Does your research involve animals?	NO
2 . 8	Does your research involve the administration of drugs, placebos or other substances to study participants?	NO
A.3 If you answer YES to any of the questions in this block, then unless you are applying to an external ethics committee or the SREC, you must apply for approval from the Computer Science Research Ethics Committee (CSREC) through Research Ethics Online - https://ethics.city.ac.uk/ Depending on the level of risk associated with your application, it may be referred to the Senate Research Ethics Committee.		<i>Delete as appropriate</i>
3 . 1	Does your research involve participants who are under the age of 18?	NO
3 . 2	Does your research involve adults who are vulnerable because of their social, psychological or medical circumstances (vulnerable adults)? <i>This includes adults with cognitive and / or learning disabilities, adults with physical disabilities and older people.</i>	NO
3 . 3	Are participants recruited because they are staff or students of City, University of London? <i>For example, students studying on a particular course or module. If yes, then approval is also required from the Head of Department or Programme Director.</i>	NO
3 . 4	Does your research involve intentional deception of participants?	NO
3 . 5	Does your research involve participants taking part without their informed consent?	NO
3 . 5	Is the risk posed to participants greater than that in normal working life?	NO
3 . 7	Is the risk posed to you, the researcher(s), greater than that in normal working life?	NO

	<p>A.4 If you answer YES to the following question and your answers to all other questions in sections A1, A2 and A3 are NO, then your project is deemed to be of MINIMAL RISK.</p> <p>If this is the case, then you can apply for approval through your supervisor under PROPORTIONATE REVIEW. You do so by completing PART B of this form.</p> <p>If you have answered NO to all questions on this form, then your project does not require ethical approval. You should submit and retain this form as evidence of this.</p>	<p><i>Delete as appropriate</i></p>
4	<p>Does your project involve human participants or their identifiable personal data?</p> <p><i>For example, as interviewees, respondents to a survey or participants in testing.</i></p>	<p>NO</p>

Appendix B: Code for the Best Performing Model

This appendix contains the code for the best-performing model developed in this project. The code includes all major components: dataset loading and pre-processing, the feature extraction process, model training, testing, evaluation, and real-time testing. It demonstrates how the extracted features were utilized to train the model, as well as how the model was integrated with a webcam interface for real-time gesture recognition and evaluation.

```
1. import os
2. from tqdm import tqdm
3. import cv2
4. import matplotlib.pyplot as plt
5. import random
6.
7.
8. from torchvision import datasets, transforms
9. from torch.utils.data import DataLoader
10.
11. import numpy as np
12. import torchvision
13.
14. import torch
15. from skimage import feature, color
16. from skimage.transform import resize
17. from skimage.feature import hog
18.
19. from sklearn import svm
20. import time
21. from sklearn.metrics import classification_report, accuracy_score
22.
23. from PIL import Image, ImageEnhance
24. from io import BytesIO
25.
26. ## Loading Dataset
27. # Define the root directory where the dataset is stored
28. root_dir = "data" # Update with your dataset path
29.
30. # Define transformations for the images with additional augmentations
31. transform = transforms.Compose([
32.     transforms.Resize((128, 128)), # Resize all images to a consistent size
33.     transforms.RandomHorizontalFlip(), # Randomly flip the image horizontally
34.     transforms.ToTensor(), # Convert image to tensor
35. ])
36.
37. # Load the training and testing datasets
38. train_dir = os.path.join(root_dir, "train")
39. train_dataset = datasets.ImageFolder(root=train_dir, transform=transform)
40.
41. # Create DataLoader objects to iterate through the dataset
42. train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
43.
44. # Get the class names from the dataset
45. class_names = train_dataset.classes
46. print("Classes:", class_names)
47.
48. # Example of accessing images and labels from the train_loader
49. for images, labels in train_loader:
50.     print(f"Batch of images shape: {images.shape}")
51.     print(f"Batch of labels: {labels}")
52.     break # Remove this to go through the entire dataset
```

```

53.
54. ## Model 5: Mediapipe angle+finger curl+palm orientation feature Extraction
55.
56. import mediapipe as mp
57. import numpy as np
58. import torch
59. from tqdm import tqdm
60. from PIL import Image
61.
62. # Initialize MediaPipe Hands
63. mp_hands = mp.solutions.hands
64. hands = mp_hands.Hands(static_image_mode=True, max_num_hands=1,
min_detection_confidence=0.5)
65. mp_drawing = mp.solutions.drawing_utils
66.
67. # Helper function to calculate angles between vectors
68. def calculate_angle(pointA, pointB, pointC):
69.     """
70.     Calculate the angle at pointB formed by vectors BA and BC.
71.     """
72.     BA = np.array(pointA) - np.array(pointB)
73.     BC = np.array(pointC) - np.array(pointB)
74.     dot_product = np.dot(BA, BC)
75.     magnitude_BA = np.linalg.norm(BA)
76.     magnitude_BC = np.linalg.norm(BC)
77.     cos_theta = dot_product / (magnitude_BA * magnitude_BC)
78.     angle = np.arccos(np.clip(cos_theta, -1.0, 1.0)) # Clip to avoid numerical issues
79.     return np.degrees(angle)
80.
81. # Helper function to calculate finger curl
82. def calculate_curl(base, pip, tip):
83.     """
84.     Calculate the curl of a finger.
85.     base: MCP joint of the finger
86.     pip: PIP joint of the finger
87.     tip: Tip of the finger
88.     Curl is calculated as the ratio of the distances: (base to tip) / (base to pip)
89.     """
90.     distance_base_to_tip = np.linalg.norm(np.array(tip) - np.array(base))
91.     distance_base_to_pip = np.linalg.norm(np.array(pip) - np.array(base))
92.     return distance_base_to_tip / distance_base_to_pip
93.
94. # Helper function to calculate palm orientation
95. def calculate_palm_orientation(wrist, middle_mcp, pinky_mcp):
96.     """
97.     Calculate the orientation of the palm using vectors.
98.     wrist: Wrist landmark (keypoints[0])
99.     middle_mcp: Middle finger MCP joint (keypoints[9])
100.    pinky_mcp: Pinky finger MCP joint (keypoints[17])
101.    Returns roll, pitch, yaw angles in degrees.
102.    """
103.    # Create vectors
104.    wrist_to_middle = np.array(middle_mcp) - np.array(wrist)
105.    wrist_to_pinky = np.array(pinky_mcp) - np.array(wrist)
106.
107.    # Normal vector to the palm plane (cross product)
108.    palm_normal = np.cross(wrist_to_middle, wrist_to_pinky)
109.    palm_normal /= np.linalg.norm(palm_normal) # Normalize
110.
111.    # Calculate roll (rotation around x-axis)
112.    roll = np.arctan2(palm_normal[1], palm_normal[2])
113.
114.    # Calculate pitch (rotation around y-axis)
115.    pitch = np.arctan2(-palm_normal[0], np.sqrt(palm_normal[1]**2 + palm_normal[2]**2))
116.
117.    # Calculate yaw (rotation around z-axis)
118.    yaw = np.arctan2(wrist_to_middle[1], wrist_to_middle[0])
119.
120.    # Convert to degrees
121.    roll = np.degrees(roll)

```

```

122.     pitch = np.degrees(pitch)
123.     yaw = np.degrees(yaw)
124.
125.     return roll, pitch, yaw
126.
127. # Initialize storage for features and labels
128. features = [] # List of arrays, where each array is the feature vector for one image
129. labels = [] # List of corresponding labels
130.
131. # Iterate through DataLoader with tqdm for progress tracking
132. for images, batch_labels in tqdm(train_loader, desc="Processing Images",
total=len(train_loader)):
133.     for image, label in zip(images, batch_labels):
134.         # Convert tensor to PIL image
135.         pil_image = transforms.ToPILImage()(image)
136.
137.         # Prepare image for MediaPipe (convert to RGB)
138.         rgb_image = np.array(pil_image.convert("RGB"))
139.
140.         # Perform hand pose estimation
141.         results = hands.process(rgb_image)
142.
143.         if results.multi_hand_landmarks:
144.             for hand_landmarks in results.multi_hand_landmarks:
145.                 # Extract keypoints
146.                 keypoints = [
147.                     [lm.x * pil_image.width, lm.y * pil_image.height, lm.z]
148.                     for lm in hand_landmarks.landmark
149.                 ]
150.
151.                 # Compute angles for all fingers
152.                 finger_angles = []
153.
154.                 # Index finger
155.                 finger_angles.append(calculate_angle(keypoints[5], keypoints[6],
keypoints[7])) # MCP joint
156.                 finger_angles.append(calculate_angle(keypoints[6], keypoints[7],
keypoints[8])) # PIP joint
157.
158.                 # Middle finger
159.                 finger_angles.append(calculate_angle(keypoints[9], keypoints[10],
keypoints[11])) # MCP joint
160.                 finger_angles.append(calculate_angle(keypoints[10], keypoints[11],
keypoints[12])) # PIP joint
161.
162.                 # Ring finger
163.                 finger_angles.append(calculate_angle(keypoints[13], keypoints[14],
keypoints[15])) # MCP joint
164.                 finger_angles.append(calculate_angle(keypoints[14], keypoints[15],
keypoints[16])) # PIP joint
165.
166.                 # Pinky finger
167.                 finger_angles.append(calculate_angle(keypoints[17], keypoints[18],
keypoints[19])) # MCP joint
168.                 finger_angles.append(calculate_angle(keypoints[18], keypoints[19],
keypoints[20])) # PIP joint
169.
170.                 # Thumb
171.                 finger_angles.append(calculate_angle(keypoints[0], keypoints[1],
keypoints[2])) # Base joint
172.                 finger_angles.append(calculate_angle(keypoints[1], keypoints[2],
keypoints[4])) # Tip joint
173.
174.                 # Compute inter-finger angles
175.                 inter_finger_angles = []
176.                 inter_finger_angles.append(calculate_angle(keypoints[1], keypoints[5],
keypoints[9])) # Thumb to index
177.                 inter_finger_angles.append(calculate_angle(keypoints[5], keypoints[9],
keypoints[13])) # Index to middle

```

```

178.             inter_finger_angles.append(calculate_angle(keypoints[9], keypoints[13],
keypoints[17])) # Middle to ring
179.
180.             # Compute finger curl
181.             finger_curls = []
182.             finger_curls.append(calculate_curl(keypoints[0], keypoints[1],
keypoints[4])) # Thumb curl
183.             finger_curls.append(calculate_curl(keypoints[5], keypoints[6],
keypoints[8])) # Index curl
184.             finger_curls.append(calculate_curl(keypoints[9], keypoints[10],
keypoints[12])) # Middle curl
185.             finger_curls.append(calculate_curl(keypoints[13], keypoints[14],
keypoints[16])) # Ring curl
186.             finger_curls.append(calculate_curl(keypoints[17], keypoints[18],
keypoints[20])) # Pinky curl
187.
188.             # Compute palm orientation
189.             roll, pitch, yaw = calculate_palm_orientation(
190.                 keypoints[0], keypoints[9], keypoints[17]
191.             )
192.
193.             # Combine all features into a single feature vector
194.             feature_vector = np.array(finger_angles + inter_finger_angles +
finger_curls + [roll, pitch, yaw])
195.             features.append(feature_vector)
196.             labels.append(label.item()) # Store label as integer
197.
198. # Convert features and labels to NumPy arrays for SVM training
199. features = np.array(features) # Shape: (number_of_images, number_of_features)
200. labels = np.array(labels) # Shape: (number_of_images,)
201.
202. # Print shapes for verification
203. print("Features shape:", features.shape)
204. print("Labels shape:", labels.shape)
205.
206. ## Create a list `l` that maps each label in `labels_list` to its corresponding class name
207. l=[]
208. for j in range(len(labels)):
209.     l.append(class_names[labels[j]])
210.
211. from sklearn.model_selection import train_test_split
212.
213. # Split the HOG features and labels into training and testing sets
214. X_train, X_test, y_train, y_test = train_test_split(
215.     features, l, test_size=0.2, random_state=42, stratify=labels
216. )
217.
218. # Print the shapes of the resulting datasets
219. print(f"Training set size: {X_train.shape[0]} images")
220. print(f"Testing set size: {X_test.shape[0]} images")
221.
222. #Model Training
223. svm_classifier = svm.SVC(kernel='poly',degree=6)
224.
225. start_time = time.time()
226.
227. svm_classifier.fit(X_train,y_train)
228.
229. end_time = time.time()
230. print("Elapsed time: ", end_time - start_time)
231.
232. ## Model Prediction on test set
233. y_pred = svm_classifier.predict(X_test)
234. print("Accuracy: "+str(accuracy_score(y_test, y_pred)))
235. print('\n')
236. print(classification_report(y_test, y_pred))
237.
238. import numpy as np
239. import matplotlib.pyplot as plt
240. from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

```

```

241.
242. # Find the unique labels in both testLabels and y_pred
243. all_labels = np.unique(np.concatenate([y_test, y_pred]))
244.
245. # Compute the confusion matrix
246. cm = confusion_matrix(y_test, y_pred, labels=all_labels)
247.
248. # Create a large figure and plot the confusion matrix
249. fig, ax = plt.subplots(figsize=(12, 12)) # Set the desired figure size
250.
251. # Create confusion matrix display
252. disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=all_labels)
253.
254. # Plot with rotated x-axis labels to avoid overlap
255. disp.plot(cmap=plt.cm.Blues, ax=ax, xticks_rotation='vertical')
256.
257. # Set a title
258. plt.title('Confusion Matrix')
259. plt.show()
260.
261. ## Real-Time testing
262. import cv2
263. import mediapipe as mp
264. import numpy as np
265. import pickle # To load the trained SVM model
266.
267. # Load local images
268. background_img = cv2.imread('bg.jpg') # Replace with your background image path
269. title_img = cv2.imread('title.jpg') # Replace with your title image path
270.
271. # Resize the title image to fit the top of the UI
272. title_img = cv2.resize(title_img, (600, 100)) # Adjust the dimensions as needed
273.
274. # Initialize MediaPipe Hands
275. mp_hands = mp.solutions.hands
276. hands = mp_hands.Hands(min_detection_confidence=0.7, min_tracking_confidence=0.5)
277.
278. # Function to calculate angle between vectors
279. def calculate_angle(pointA, pointB, pointC):
280.     BA = np.array(pointA) - np.array(pointB)
281.     BC = np.array(pointC) - np.array(pointB)
282.     dot_product = np.dot(BA, BC)
283.     magnitude_BA = np.linalg.norm(BA)
284.     magnitude_BC = np.linalg.norm(BC)
285.     cos_theta = dot_product / (magnitude_BA * magnitude_BC)
286.     angle = np.arccos(np.clip(cos_theta, -1.0, 1.0))
287.     return np.degrees(angle)
288.
289. # Function to calculate finger curl
290. def calculate_curl(base, pip, tip):
291.     distance_base_to_tip = np.linalg.norm(np.array(tip) - np.array(base))
292.     distance_base_to_pip = np.linalg.norm(np.array(pip) - np.array(base))
293.     return distance_base_to_tip / distance_base_to_pip
294.
295. # Function to calculate palm orientation (roll, pitch, yaw)
296. def calculate_palm_orientation(wrist, middle_mcp, pinky_mcp):
297.     wrist_to_middle = np.array(middle_mcp) - np.array(wrist)
298.     wrist_to_pinky = np.array(pinky_mcp) - np.array(wrist)
299.
300.     palm_normal = np.cross(wrist_to_middle, wrist_to_pinky)
301.     palm_normal /= np.linalg.norm(palm_normal)
302.
303.     roll = np.arctan2(palm_normal[1], palm_normal[2])
304.     pitch = np.arctan2(-palm_normal[0], np.sqrt(palm_normal[1]**2 + palm_normal[2]**2))
305.     yaw = np.arctan2(wrist_to_middle[1], wrist_to_middle[0])
306.
307.     return np.degrees(roll), np.degrees(pitch), np.degrees(yaw)
308.
309. # Start capturing video from the webcam
310. cap = cv2.VideoCapture(0)

```

```

311.
312. if not cap.isOpened():
313.     print("Error: Could not open video stream.")
314.     exit()
315.
316. # Define padding and transparency
317. padding = 20
318. overlay_alpha = 0.6
319.
320. # Gap between title and live feed
321. title_gap = 50 # Adjust the gap as needed
322.
323. while True:
324.     # Read each frame from the webcam
325.     ret, frame = cap.read()
326.     if not ret:
327.         print("Error: Failed to capture image")
328.         break
329.
330.     # Flip the frame horizontally for a later selfie-view display
331.     frame = cv2.flip(frame, 1)
332.
333.     # Convert the BGR image to RGB
334.     rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
335.
336.     # Process the frame and detect hands
337.     results = hands.process(rgb_frame)
338.
339.     # Resize background to match webcam frame
340.     background_resized = cv2.resize(background_img, (frame.shape[1], frame.shape[0]))
341.
342.     # Dimensions for components
343.     title_width = 600
344.     title_height = 100
345.     live_feed_width = 900
346.     live_feed_height = 480
347.     prediction_width = 600
348.     prediction_height = 100
349.
350.     # Calculate the center positions
351.     title_x = (frame.shape[1] - title_width) // 2
352.     title_y = 20 # Place the title at the top
353.     live_feed_x = (frame.shape[1] - live_feed_width) // 2
354.     live_feed_y = title_y + title_height + title_gap # Centered with a gap below the title
355.     prediction_x = (frame.shape[1] - prediction_width) // 2
356.     prediction_y = frame.shape[0] - prediction_height - 20 # Centered at the bottom
357.
358.     # Place the title at the top
359.     background_resized[title_y:title_y + title_img.shape[0],
360.                        title_x:title_x + title_img.shape[1]] = title_img
361.
362.     # Place the live feed in a box
363.     frame_resized = cv2.resize(frame, (live_feed_width, live_feed_height))
364.     background_resized[live_feed_y:live_feed_y + live_feed_height,
365.                        live_feed_x:live_feed_x + live_feed_width] = frame_resized
366.
367.     # Initialize prediction text
368.     prediction_text = ""
369.
370.     if results.multi_hand_landmarks:
371.         for hand_landmarks in results.multi_hand_landmarks:
372.             # Get the hand coordinates
373.             h, w, _ = frame.shape
374.             keypoints = [
375.                 [landmark.x * w, landmark.y * h, landmark.z]
376.                 for landmark in hand_landmarks.landmark
377.             ]
378.
379.             # Compute angles for all fingers
380.             finger_angles = []

```

```

381.         finger_angles.append(calculate_angle(keypoints[5], keypoints[6], keypoints[7]))
# Index MCP
382.         finger_angles.append(calculate_angle(keypoints[6], keypoints[7], keypoints[8]))
# Index PIP
383.         finger_angles.append(calculate_angle(keypoints[9], keypoints[10],
keypoints[11])) # Middle MCP
384.         finger_angles.append(calculate_angle(keypoints[10], keypoints[11],
keypoints[12])) # Middle PIP
385.         finger_angles.append(calculate_angle(keypoints[13], keypoints[14],
keypoints[15])) # Ring MCP
386.         finger_angles.append(calculate_angle(keypoints[14], keypoints[15],
keypoints[16])) # Ring PIP
387.         finger_angles.append(calculate_angle(keypoints[17], keypoints[18],
keypoints[19])) # Pinky MCP
388.         finger_angles.append(calculate_angle(keypoints[18], keypoints[19],
keypoints[20])) # Pinky PIP
389.         finger_angles.append(calculate_angle(keypoints[0], keypoints[1], keypoints[2]))
# Thumb base
390.         finger_angles.append(calculate_angle(keypoints[1], keypoints[2], keypoints[4]))
# Thumb tip
391.
392.         # Compute inter-finger angles
393.         inter_finger_angles = []
394.         inter_finger_angles.append(calculate_angle(keypoints[1], keypoints[5],
keypoints[9])) # Thumb-Index
395.         inter_finger_angles.append(calculate_angle(keypoints[5], keypoints[9],
keypoints[13])) # Index-Middle
396.         inter_finger_angles.append(calculate_angle(keypoints[9], keypoints[13],
keypoints[17])) # Middle-Ring
397.
398.         # Compute finger curl
399.         finger_curls = []
400.         finger_curls.append(calculate_curl(keypoints[0], keypoints[1], keypoints[4]))
# Thumb curl
401.         finger_curls.append(calculate_curl(keypoints[5], keypoints[6], keypoints[8]))
# Index curl
402.         finger_curls.append(calculate_curl(keypoints[9], keypoints[10], keypoints[12]))
# Middle curl
403.         finger_curls.append(calculate_curl(keypoints[13], keypoints[14],
keypoints[16])) # Ring curl
404.         finger_curls.append(calculate_curl(keypoints[17], keypoints[18],
keypoints[20])) # Pinky curl
405.
406.         # Compute palm orientation
407.         roll, pitch, yaw = calculate_palm_orientation(
408.             keypoints[0], keypoints[9], keypoints[17]
409.         )
410.
411.         # Combine all features into a single feature vector
412.         feature_vector = np.array(finger_angles + inter_finger_angles + finger_curls +
[roll, pitch, yaw])
413.
414.         # Make a prediction using the SVM classifier
415.         prediction = svm_classifier.predict([feature_vector])[0]
416.         prediction_text = f"Prediction: {prediction}"
417.
418.         # Draw a transparent rounded box for prediction
419.         overlay = background_resized.copy()
420.         cv2.rectangle(
421.             overlay,
422.             (prediction_x, prediction_y),
423.             (prediction_x + prediction_width, prediction_y + prediction_height),
424.             (0, 0, 0),
425.             -1,
426.         )
427.         cv2.addWeighted(overlay, overlay_alpha, background_resized, 1 - overlay_alpha, 0,
background_resized)
428.
429.         # Add prediction text inside the box
430.         cv2.putText(

```



```

431.         background_resized,
432.         prediction_text,
433.         (prediction_x + 30, prediction_y + 60),
434.         cv2.FONT_HERSHEY_SIMPLEX,
435.         1,
436.         (255, 255, 255),
437.         2,
438.         cv2.LINE_AA,
439.     )
440.
441.
442.
443.     # Display the frame with everything
444.     cv2.imshow('Hand Detection', background_resized)
445.
446.     # Press 'q' to exit the webcam window
447.     if cv2.waitKey(1) & 0xFF == ord('q'):
448.         break
449.
450. # Release the webcam and close all OpenCV windows
451. cap.release()
452. cv2.destroyAllWindows()

```

Appendix C: Prompts Used with ChatGPT

The following prompts were used to assist in refining and improving the clarity, conciseness, and formalization of the report:

Prompt 1: "Help me reword the following paragraph for clarity and flow?"

Prompt 2: "Make this paragraph concise without losing the key information."

Prompt 3: "Help me rephrase this conclusion to make it more impactful."

Prompt 4: "Check this section for grammatical errors and suggest corrections."

Prompt 5: "Formalise the following paragraph for academic writing."

Appendix D: Real-Time Testing Screenshots

Below are additional screenshots showcasing the real-time testing of the best-performing model under various conditions. These images demonstrate the model's performance in recognizing sign language gestures across different lighting, backgrounds, and hand positions, and both left and right hands, highlighting its robustness and accuracy in real-world scenarios.

