OCR A-Level Computer Science H446 NEA

Sign Language Fingerspelling Recognition System

Full Name: Mahe Chowdury
Candidate Number: 9238
Centre Name: Six 21
Centre Number: 13284

Contents

Analysis

Before initiating development, it is crucial to identify and comprehend the problem(s) and the needs of stakeholders to propose appropriate solutions.

Problem Identification

Introduction

Verbal communication is defined as the act of sharing and exchanging information. A shared language and mutual comprehension are required for effective communication between individuals. Communication alternatives for those who are deaf and unable to talk, on the other hand, differ. Deafness is the inability to hear, while muteness is the inability to speak. Deaf and mute people use sign language to communicate with each other and with those who can hear. Unfortunately, persons who do not have such limitations frequently miss the importance of sign language.

Fingerspelling is a component of sign language used to spell out words by employing specific hand gestures and movements. It serves as a method of representing written text within sign language, typically when no specific sign exists for a particular word. This is especially common when spelling proper nouns, names, and technical terms [1]. Each of the 26 letters in the alphabet corresponds to a distinct hand sign. For example, in American Sign Language (ASL), 'A' is depicted as a closed fist with the thumb extended. It's important to note that these hand signs may vary across countries, depending on the specific sign language used. In the UK, for instance, British Sign Language (BSL) is prevalent, while in India, the Indian Sign Language system (ISL) is utilised. As an integral part of sign language, fingerspelling can pose its own set of challenges.

Problem Identification

After conducting extensive research, I identified that even with the existence of numerous sign languages, the overall percentage of the population proficient in any of them remains low. This poses a challenge for individuals with special needs, hindering their ability to communicate seamlessly with others. Sign language recognition serves as a solution, enabling communication in sign language without the necessity of learning it. The technology identifies gestures and translates them into widely spoken languages such as English.There is a need for software that can translate these sign languages into text, which can serve several valuable purposes:

1. **Enhancing Communication:** This software allows individuals to fingerspell in real-time and receive immediate output in the form of corresponding letters, which can be combined to form words. This capability greatly facilitates communication with deaf individuals, bridging the communication gap. Moreover, individuals with hearing impairments can also utilise this tool to communicate effectively with both the deaf and hearing communities.
    - Despite sign language being more effective than fingerspelling for conveying complex ideas, sentences, and emotions because it involves a full range of hand movements, facial expressions, and body language, developing software for sign language recognition poses several challenges:
        - **Complexity:** Sign language is complex due to its wide range of signs, gestures, facial expressions, and body movements, making recognition and interpretation challenging.
        - **Vocabulary Size:** Sign language features an extensive vocabulary with thousands of distinct signs, in contrast to fingerspelling, which consists of 26 letters.
        - **Data Collection:** Collecting and annotating data for fingerspelling recognition is relatively straightforward compared to creating an extensive dataset for full sign language recognition, which would necessitate a significantly larger and more diverse set of signs.
2. **Independence:** Deaf individuals can use this software to communicate more independently, reducing their dependency on interpreters or family members for translation.
3. **Educational Support:** It can serve as a valuable educational tool for individuals learning sign language, especially for beginners who may encounter difficulties in interpreting fingerspelling. Real-time feedback can significantly aid in the learning process.

Computational Methods

Addressing this challenge relies on the application of computational methods to discover and execute an appropriate solution. Utilising a webcam is essential for detecting hand gestures and movements. This capability will be accessible on a computer because the real-time recognition of fingerspelling gestures demands substantial computational resources, especially when aiming to provide instant feedback.

Problem Recognition

The conversion of complex hand movements into precise letters of the alphabet is the main difficulty in building fingerspelling recognition software. Further investigation reveals that the primary obstacle is the precise identification of a hand within a given image or video frame. After overcoming this obstacle, the algorithm will proceed to collect important information about the hand's location and arrangement in order to understand the fingerspelling gestures. This includes not only recognising the presence of a hand, but also modifying it in such a way that complicated actions may be recognised.

Problem Decomposition

This problem can be broken down into a set of smaller and simpler steps. Here's an initial outline of the steps my software will follow in order to function:

Algorithm: Recognition.
Input: Fingerspelling Image.
Output: Letter prediction of input image.
Method: Step 1: Capture image through camera/webcam.
Step 2: Locate position of hand in the image, using a detector.
Step 3: Transform image (apply necessary filters/modifications to outline shape of hand).
Step 4: Classify the object, extracting necessary features needed to recognise the associated letter.
Step 5: Display the letter.
Recognition ends [2].

This needs to be done in real time, providing the user with an immediate result. With computer vision, the collection and modification of the input can be accomplished swiftly. It is the recognition part that will demand significant computational power; hence, the model needs to be well-trained to ensure a prompt response.

Divide and Conquer

Applying the divide and conquer approach involves breaking down the complex problem of fingerspelling recognition into smaller, manageable steps. By addressing these steps individually and refining them as separate modules, we can focus on specific aspects like hand detection and gesture classification. Once these components are optimised, they can be integrated into the larger software framework, aligning with the divide and conquer method for more structured development.

Abstraction

Abstraction in fingerspelling recognition software streamlines the recognition process, reduces computational complexity, and enhances the software's efficiency by prioritising the essential aspects of hand gestures and their interpretation. This simplification is crucial for achieving real-time and accurate fingerspelling recognition.

Data reduction and feature extraction are important areas where abstraction plays a crucial role in fingerspelling recognition software. Firstly, the software's abstraction focuses on isolating the hand gesture from the background, effectively eliminating unnecessary data. Additionally, adjusting brightness levels can enhance the image quality, thereby improving recognition accuracy. Furthermore, the idea of converting the image to grayscale, potentially by applying a mask, simplifies processing by reducing the image to simple information.

Stakeholders

The primary users are individuals with hearing impairments, primarily the deaf and hard-of-hearing community. They seek a tool to bridge the gap between sign language and spoken or written language, facilitating communication with the general public.

Stakeholders include individuals of diverse ages, offering an opportunity to collect a representative sample that mimics the wide-ranging communication requirements within the deaf community. It's essential to acknowledge that each age group within the community may have distinct needs, focusing on the importance of effectively addressing these unique demands.

- **Deaf Children/Teens (Age: 13-17)**: They require appropriate communication tools to facilitate language development, academic progress, and interpersonal relationships. Sign language recognition software can be used by children to practise communication with peers, teachers, and family members.
- **Deaf Adults/Elders (Age: 18-65):** This demographic may require sign language recognition software for various purposes, such as social engagement, accessing services, and professional interactions. The program can help individuals overcome communication barriers and actively participate in both social and professional contexts.

Aspiring Sign Language Learners: This is another type of stakeholders that I decided to include. Whether they want to become interpreters and communication specialists or simply want to communicate better with the deaf community, are another major group of stakeholders. These people appreciate the importance of mastering hand gestures and try to improve their fingerspelling abilities. To meet their needs, it is essential to include a system within the sign language recognition software that provides real-time feedback. This feedback system may involve displaying the accuracy of fingerspelling gestures, allowing learners to analyse their progress and make required corrections. By including this feature, the software allows users to enhance their signing abilities, hence improving their overall proficiency in sign language communication.

I have selected one representative from each of the categories mentioned earlier to be my stakeholders. I have opted for email communication, considering their busy schedules, as it allows them the flexibility to respond to my enquiries at their convenience.

Name	Yusuf I. (Age: 17)	Robert D. (Age: 28)	Imtiaz R. (Age: 20)
Category	Deaf Teen	Deaf Adult	Learner
Description	A deaf teenager who is 17 years old is eager to develop his fingerspelling skills. He strives to become more fluent in his signing because he understands the importance of clear communication in the deaf community. Yusuf views the sign language recognition software as a useful tool that will enable him to improve his signing skills and interact with others more successfully in social and educational contexts.	Robert is a professional working in a busy office environment who relies on sign language for communication, despite having little hearing impairment. He recognises the challenges he faces in a work environment when communicating with other colleagues (especially new individuals). He hopes to improve his capacity to communicate effectively with coworkers, participate actively in meetings, and fully engage in his professional responsibilities by using the software.	Imtiaz is an ordinary person who feels compelled to learn sign language personally since his brother is deaf. Because he understands how crucial it is to have a close relationship, he wants to be able to communicate with his brother more successfully. He recognises the value of sign language recognition software as a learning tool that will aid in the development of the abilities required to conduct effective discussions and strengthen his sibling bond.

Interview/Observation Groups

In order to get their valuable input and feedback on the creation of my fingerspelling recognition program, I will pose a specific set of questions to the stakeholders during the interviews, demanding a very detailed response, so that I can give them the best experience when using my software. Thus, I sent them this message and gave them time to think thoroughly. While I will utilise these questions as a starting point, I am ready to investigate more and make further requests if needed. These questions' main goal is to learn more about the stakeholders' thoughts on the software, its potential applications, and how they see themselves using it to meet their particular needs.

Interview Questions

Questions - Deaf Individual

The following are the questions I have for Adam and Robert (who represent the deaf individuals):

1. *What are the main problems or difficulties you have when it comes to fingerspelling or sign language communication?*
2. *What features or functionalities in a fingerspelling software would you like to see to improve your communication with other people or learning experience?*
3. *How do you intend to use the fingerspelling software in your interactions with others?*
4. *Do you have any particular fingerspelling patterns, gestures, or parts of sign language that you find particularly difficult to gesture?*
5. *If so, how do you think the program can help you improve your understanding and performance of these gestures?*
6. *What other ways do you believe fingerspelling software can help you or the deaf community as a whole?*
7. *Is there anything else you'd want to add or suggest that could assist in the building of an efficient fingerspelling program?*

The primary objectives of questions 1 and 3 are to acquaint me with the stakeholders, allowing me to understand their personalities through how they envision using my software and their prior experiences with fingerspelling. This insight into their preferences is essential for tailoring the software to meet their needs effectively.

I aim to discover how they intend to integrate my software into their daily interactions, be it for personal communication or work-related tasks. This information is invaluable for ensuring the software's versatility to accommodate various use cases, thus I decided to ask question 2.

Additionally, questions 4, and 5 serve the purpose of identifying specific challenges related to sign language or fingerspelling patterns. This information guides the development of targeted solutions aimed at enhancing their signing skills.

As for questions 6 and 7, while not strictly necessary, they provide an opportunity to explore potential benefits for other individuals and gather additional suggestions, insights, and ideas.

Questions - Learner

The following are the questions I have for Imtiaz (who represents the learners):

1. *How well do you know sign language and fingerspelling? Do you have any prior experience or are you just starting out?*
2. *What inspired you to learn sign language and practise fingerspelling?*
3. *Have you faced any difficulties or obstacles while learning to spell with your fingers?*
4. *If yes, what were they, and how do you believe a fingerspelling recognition software might help with solving them?*
5. *How important do you think real-time feedback is for improving your fingerspelling skills?*
6. *How often and in what settings do you expect to use the fingerspelling program?*

7. *Is there anything else you'd want to add or suggest that could assist in the building of an efficient fingerspelling program?*

Questions 1-4 are aimed at understanding the stakeholder and identifying potential scenarios where fingerspelling may be used. This information is invaluable for recognising challenges stakeholders face in their daily communication. An in-depth understanding of these difficulties is vital for crafting features that effectively address specific issues and align with stakeholder preferences.

Question 6 focuses on exploring potential benefits beyond personal use, allowing for the consideration of broader implications. Stakeholders may propose innovative applications or functionalities that enhance the software's overall utility for the deaf community.

Question 7, being an open-ended enquiry, encourages stakeholders to share additional insights, ideas, or suggestions. They might suggest features or improvements that have not been previously considered, thereby enriching the software development process.

Interview Answers

Below are the stakeholders' responses to my questions. These will be used to understand their needs and wants, in order to define the main system requirements.

Answers - Deaf Individual (Yusuf and Robert)	
1. *What are the main problems or difficulties you have when it comes to fingerspelling or sign language communication?*	
Yusuf	*"I sometimes struggle with getting the right handshapes and movements in fingerspelling and sign language communication. It would be great to have a recognition software to help improve my accuracy and fluency. Additionally, in noisy environments, the software could assist me in following conversations more effectively."*
Robert	*"One of the most difficult aspects is ensuring that my signing is clear and understandable, particularly when engaging with colleagues who may be unfamiliar with sign language."*
2. *What features or functionalities in a fingerspelling software would you like to see to improve your communication with other people or learning experience?*	
Yusuf	*"I'd like a software that offers real-time feedback on my signing accuracy and provides interactive exercises to enhance my learning experience."*
Robert	*"All I need is a fingerspelling program with real-time translation to assist me with communicating."*
3. *How do you intend to use the fingerspelling software in your interactions with others?*	
Yusuf	*"I'll use the fingerspelling software to improve my signing skills and communicate better with others."*
Robert	*"The real-time translation option would be extremely useful when communicating with staff members who are not fluent in sign language."*
4. *Do you have any particular fingerspelling patterns, gestures, or parts of sign language that you find particularly difficult to gesture?*	
Yusuf	*"Sometimes, I find fingerspelling longer words or complex signs challenging, especially when it comes to maintaining speed and accuracy."*
Robert	*"No."*

5.	*If so, how do you think the program can help you improve your understanding and performance of these gestures?*
Yusuf	*"I believe the program can help by providing real-time feedback and suggestions for improvement. It could offer interactive exercises and repetition to reinforce difficult gestures, which would boost my understanding and performance over time."*
6.	*What other ways do you believe fingerspelling software can help you or the deaf community as a whole?*
Yusuf	*"Not sure yet."*
Robert	*"I feel that including fingerspelling software into virtual meetings would be extremely beneficial, allowing real-time translation of signing during virtual discussions, ensuring that all participants, regardless of signing abilities, may effectively understand and contribute. In professional environments, this could support a more inclusive and collaborative workplace."*
7.	*Is there anything else you'd want to add or suggest that could assist in the building of an efficient fingerspelling program?*
Yusuf	*"Hmm, having a user-friendly interface with clear visual cues for handshapes and movements would make the learning process smoother."*
Robert	*"No."*

Answers - Learner (Imtiaz)	
1.	*How well do you know sign language and fingerspelling? Do you have any prior experience or are you just starting out?*
Imtiaz	*"I'm still in the learning phase, but I'm keen to pick up sign language and fingerspelling skills."*
2.	*What inspired you to learn sign language and practise fingerspelling?*
Imtiaz	*"My main inspiration is my brother, who is deaf. I really want to connect with him on a deeper level and have meaningful conversations. Learning sign language and fingerspelling seems like the perfect way to bridge the communication gap and strengthen our sibling bond!"*
3.	*Have you faced any difficulties or obstacles while learning to spell with your fingers?*
Imtiaz	*"Oh yeah, there have been some challenges along the way."*
4.	*If yes, what were they, and how do you believe a fingerspelling recognition software might help with solving them?*
Imtiaz	*"Getting the handshapes right and remembering all the letters can be a bit tricky sometimes. But I think a fingerspelling recognition software could be a game-changer. It could give me real-time feedback and correct my handshapes, helping me improve accuracy."*
5.	*How important do you think real-time feedback is for improving your fingerspelling skills?*
Imtiaz	*"Real-time feedback is crucial for me. It helps me correct mistakes and learn faster while practising fingerspelling."*
6.	*How often and in what settings do you expect to use the fingerspelling program?*
Imtiaz	*"I'd use the fingerspelling program regularly, probably a few times a week."*

	7. Is there anything else you'd want to add or suggest that could assist in the building of an efficient fingerspelling program?
Imtiaz	*"Hmm, one thing that could be awesome is if the program could have themed challenges or games. It would make learning fingerspelling even more engaging and fun. Oh, and maybe a feature where I can track my progress and see how I'm improving over time. That would be a real motivator!"*

Stakeholders Requirements

Stakeholders prioritise real-time feedback. Proposed strategies include showing both the intended and displayed letters during fingerspelling and incorporating a percentage accuracy indication. This allows users to assess accuracy instantly, facilitating efficient communication. An enhancement suggestion is to change the user's hand colour from red to green upon correct fingerspelling, improving usability.

Requested features from stakeholders include interactive exercises to improve fingerspelling skills. These exercises can be organised into levels, with easier levels focusing on simpler letters and enabling users to practise word fingerspelling. A flashcard concept may be implemented, where previously fingerspelled words periodically reappear to reinforce learning. Practice sessions could also include a visual guide on hand positioning to aid users in accurately mimicking fingerspelling.

Robert suggested integrating this software into meeting applications like Zoom or Google Meet. This integration could be particularly useful when audio fails, and users need to fingerspell. The software could translate key information like names of people or places, enhancing accessibility during meetings without interruption. However, modifying Zoom without permission likely violates the platform's terms of service. People are typically required to agree to these terms when using the software.

Lastly, it's crucial to ensure a user-friendly interface that is appropriate for all age groups. This will not only reduce reliance on human assistance in translating fingerspelling but also serve as an efficient tool for bridging the communication gap between deaf and hearing individuals if this software is accessible to the wider public.

Existing Solutions

I will review existing or similar solutions for positive features that can be integrated and identify areas that I should either avoid or include in this project, by evaluating what makes them successful

Fingerspelling.xyz

This is a web application that uses camera technology and machine learning to help the study of American Sign Language (ASL). By providing immediate feedback on their hand movements through this user-friendly platform, people can learn and improve their fingerspelling ability and make quick and efficient corrections [3], [4].

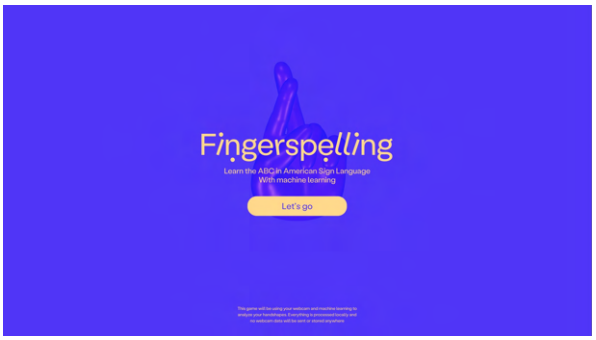


FIg. 1.1 Main Menu

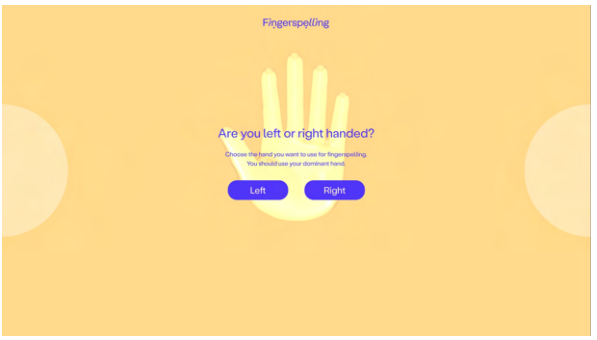


Fig. 1.2. Choose dominant hand

Fig. 1.3. Level Selection

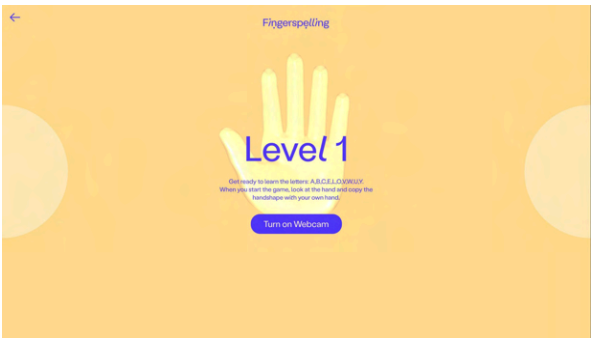


Fig. 1.4. Instructions

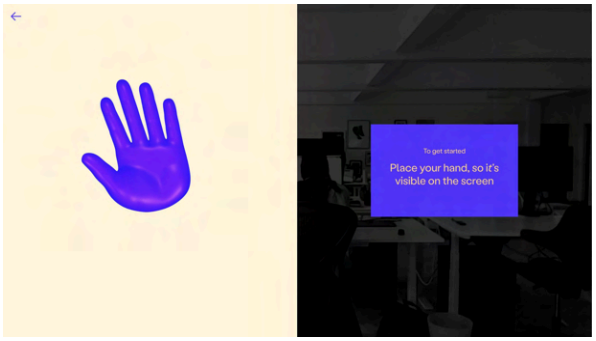


Fig. 1.5. Hand Validation

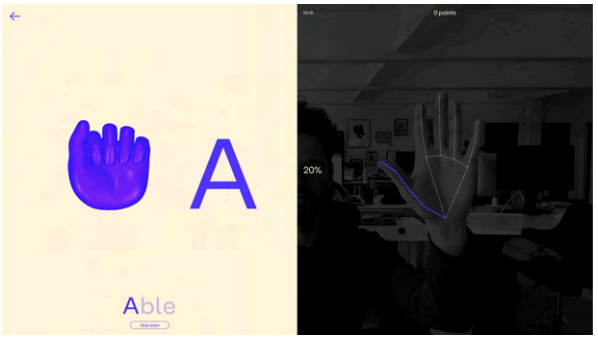


Fig. 1.6. Recognition

Fig. 1.1. [3] The welcome page's simplicity and **user-friendliness** contribute to a welcoming and inclusive environment for users of all ages. The thoughtfully chosen colour scheme, avoiding overly vibrant hues, not only enhances the page's aesthetic appeal but also ensures accessibility and ease of use. Moreover, by making the software **accessible through a webpage**, the app maximises its availability, allowing users to engage with it conveniently at any time.

Fig. 1.2. [3] **Selecting one's dominant hand** for practice is a thoughtful feature, as it demonstrates consideration for a diverse user base. This customisation option enhances user comfort and engagement with the software. Additionally, it accommodates individuals with specific physical needs, ensuring a more inclusive and improved user experience.

Fig. 1.3. [3] Incorporating a **level selection** feature is a strategic decision. It encourages users to approach fingerspelling as a progressively challenging game, where the difficulty increases as they advance through levels. Some people could be proficient in the fundamentals and want to push their limits by working through more difficult words in higher levels.

Fig. 1.4. [3] Displaying the selected level along with the letters the software will assess the user with enhances user-friendliness. This **clear and easily understandable information** engages users effectively and may make them prepare and learn in advance before starting a level, encouraging them to review and practise specific letters beforehand, contributing to a more confident and efficient learning process. This anticipatory element aligns with **adult learning principles**, where informed preparation tends to enhance engagement and retention

Fig. 1.5. [3] The recognition software initiates by ensuring the **hand is positioned correctly** in front of the camera. This step is crucial before commencing fingerspelling. The software, by guiding users to position their hands correctly, optimises the accuracy of fingerspelling interpretation, minimising errors and improving recognition reliability.

Fig. 1.6. [3] The **virtual hand** demonstrates the proper hand positioning for each letter in the displayed word below. Serving as a visual guide, it offers users a clear reference for precise letter formation. This aids users in mastering the requisite handshapes and movements crucial for fingerspelling accuracy. However, the current implementation is **present across all levels**, making advanced stages less difficult. The absence of the virtual hand guide in higher levels might make it challenging for users to remember how to fingerspell each letter in the displayed word, which would improve the learning experience. The webpage also offers **real-time feedback** by advancing to the next letter when the correct finger position matches the highlighted letter.

Ace ASL

The AI-driven solution enables individuals to practise American Sign Language (ASL). Using a camera, the system recognises signing and provides guidance [5]. The Ace ASL app utilises the sign recognition technology responsible for instant, automated translation between American Sign Language and English. This mobile application is the first ASL learning app to offer immediate signing feedback [6].

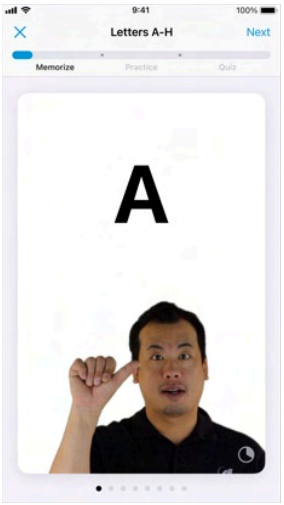


Fig. 2.1. Learning Letters

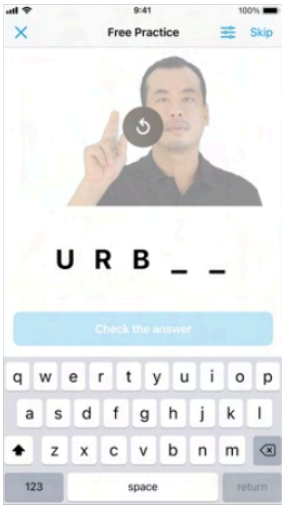


Fig. 2.2. Learning Words

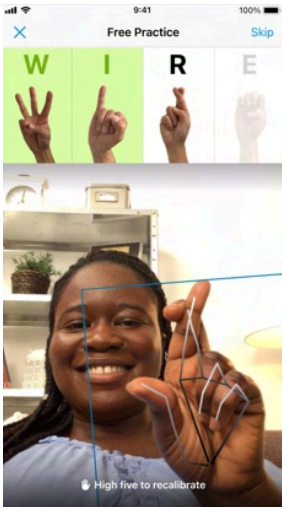


Fig. 2.3. Practice Letters

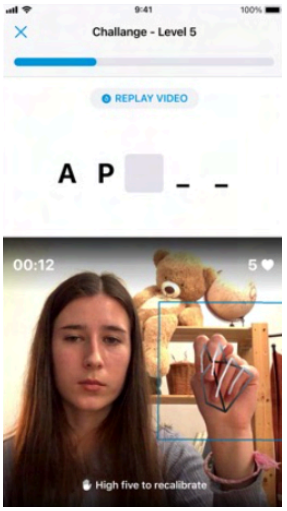


Fig. 2.4. Practice Words

Fig. 2.1. [6] In this initial image, it illustrates that users can establish a strong foundation for fingerspelling by learning hand gestures through the **displayed demonstration**. This serves as a visual reference, helping users position their fingers correctly. The **user interface** could benefit from enhancements, as it currently seems to cater primarily to older individuals, due to the monochromatic theme the GUI has. To make it more user-friendly and inclusive for people of all age groups, adjustments and improvements should be considered, like implementing some vibrant colours. Furthermore, the addition of a **progress bar** on top of the screen, serves as a tool for engagement and motivation.

Fig. 2.2. [6] Learning continues through **fingerspelling word practice**. In the image, users are tasked with recalling the letter represented by the hand gesture shown above the word. This serves as valuable practice before engaging in the app's testing phase. It appears that the user can also **play a video to see the movement of the hand** for the fingerspelling. This feature helps users in honing their fingerspelling skills by providing a comprehensive understanding of hand movements associated with each sign. The ability to play a video offers a more immersive learning experience, allowing users to grasp the difficulties of signs like J and Z (in ASL), where movement is essential to their accurate representation.

Fig. 2.3. [6] The image represents one of the testing sections provided by the app. It captures the user's finger position and uses recognition to check for a match with the labelled letter through recognition, offering **real-time feedback**. Simultaneously, it displays an image illustrating the correct finger placement, reinforcing the connection between the letter and the hand gesture, making the user very engaged. In addition, at the bottom it gives a clear instruction, when the program stops working, it allows the user to **recalibrate**, “High five to recalibrate”. Recalibrating the object recognition model through the "High five to recalibrate" instruction is an important aspect of maintaining the accuracy and effectiveness of fingerspelling practice. By initiating recalibration, users can ensure that the recognition model adapts to any changes in lighting conditions, hand positioning, or other environmental factors that may affect the accuracy of the finger position detection

Fig. 2.4. [6] In this section of the app, the user has the opportunity to **fingerspell without any guidance or assistance**, promoting a more independent and challenging learning experience. This mode not only tests the user's memory and retention of hand gestures but also encourages active recall, an essential aspect of effective learning. Users can apply the knowledge gained from previous lessons, enhancing their proficiency in fingerspelling and reinforcing the connection between signs and their corresponding letters. While this feature offers immediate feedback for self-assessment and improvement, it's important to note that the app's **exclusive availability on Google Play** [6] may limit accessibility, and consideration for expanding to other platforms could broaden its reach to a more diverse audience.

Leap Motion

Leap Motion Inc. is a company that developed a computer hardware sensor device that takes in hand gestures as input to control the computer [7]. Although this is not specifically used to learn fingerspelling, it can be used for this purpose anyways, with some little modification to the software.

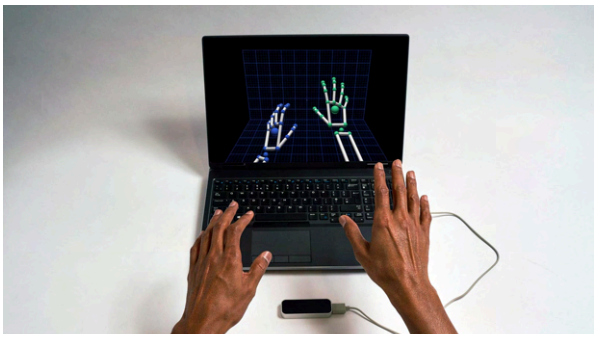


Fig. 3.1. Leap Motion in action

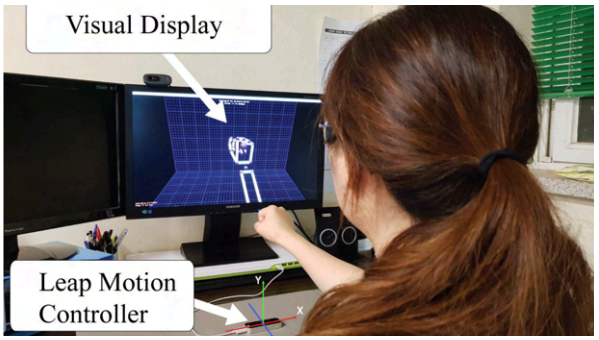


Fig. 3.2. Development of sign language recognition using Leap Motion

Fig. 3.1. [7] The controller (the black hardware piece beneath the hands) utilises infrared light for 3D hand location detection. This physical method **simplifies hand localisation and recognition**, **reducing the software's computational power** compared to visual camera-based methods. Although it's not specifically designed for fingerspelling, its primary function is depth detection for hand tracking. It's worth noting that this hardware component may be **expensive** for some users due to its essential role in recognition.

Fig. 3.2. [8] This is the only prototype I found online for using Leap Motion in the development of fingerspelling recognition software, created by T.-W.C. and B.-G.L [8]. They adopted a sensor-based approach instead of a visual one, unlike previous examples, by utilising the Leap Motion controller to capture hand gestures. However, this method comes with certain drawbacks, notably the **cost**, with the sensor itself priced at £100. They also suggested a low-cost data glove with five flex sensors on the fingers, potentially making learning sign language more expensive. Nevertheless, they later dismissed this solution, as it was often found to be intrusive and uncomfortable for users due to the bulkiness of the sensors or data glove. On the positive side, this approach results in **smoother hand recognition** compared to previous examples, mainly because data **collection through the sensor is faster and easier**. However, it **requires the hand to remain within the sensor's range**, which is a limitation for certain individuals. T.-W.C. and B.-G.L. focused primarily on developing the core recognition part of the software, with additional work needed to create the user interface and features for sign language learning.

Research Conclusion

After researching, here are the features that I can apply to my solution from:

- **Fingerspelling.xyz:** I consider this software to be one of the most successful options for learning fingerspelling. Consequently, I plan to incorporate many of its features into my own software. To begin with, the user interface is simple, a key requirement for my stakeholders. I intend to include clear and concise instructions on how to use the software. Another standout feature that I appreciate is the real-time display of the word being fingerspelled. For this aspect, I might also add an accuracy metre to indicate the precision of the hand gesture. Similarly, I plan to provide the user with guidance by displaying the correct hand position during fingerspelling.

- **Ace ASL:** What I particularly liked about the application is its level progression system. It provides a genuine challenge by requiring users to fingerspell without any hints or demonstrations, a feature that's currently lacking in fingerspelling.xyz. I intend to incorporate something similar to this in my software.

- **Leap Motion:** Since I aim to develop the fingerspelling recognition software with minimal reliance on additional hardware, I won't be incorporating features from Leap Motion. Instead, I may consider including a model of a hand on the screen or something similar.

I have also forwarded the proposed features via email to my stakeholders to gather their insights and preferences. This step ensures that we consider their valuable perspectives in the development process, and I am committed to making adjustments based on their feedback to ensure the software meets their requirements and expectations.

Stakeholder Feedback	
"What do you think of these existing solutions and the possible features to include in my program, extracted from them?"	
Yusuf	*"This seems alright. I agree with you that fingerspelling.xyz is very user friendly, which is why I find myself using it multiple times during my free time. I'm fine with everything else. Perhaps you could include a score or ranking system so that it is more enjoyable to use."* *"I will not include this feature, as my software is mainly targeted towards a wide range of people, which means some individuals may not enjoy this feature. Thank you for your feedback."*
Robert	*"Although I haven't personally utilised any of these software solutions due to not having the necessity, I believe they could prove beneficial for individuals facing similar challenges."* *"Yes I see, since you're an adult and know most of the signs in sign language you may not use my software compared to people who struggle with it. However your feedback is necessary to ensure that it is engaging with individuals with hearing impairments. Thank you for your feedback."*
Imtiaz	*"Oh I remember using fingerspelling.xyz and Ace ASL and my experience was alright, I didn't find myself struggling using them. This proposed solution for the software includes everything I need."* *"Thank you for your feedback, if you require anything else let me know."*

Proposed Solutions

Software used to recognise ASL sign language in real-time, used to learn fingerspelling, through an integrated learning environment. It will test the users in his singing ability by asking them to fingerspell certain letters.

Essential Features

These are the main features that are necessary to fulfil the functionality of my software, which I identified through the research on my stakeholders, and existing solutions.

Feature	Computational Solution
Real-time Feedback	I am going to use computer vision algorithms to process video data from the camera/webcam in real-time and track the hand. Machine learning models are crucial for the recognition part of fingerspelling translation. Additionally, efficient data transmission protocols are necessary to send recognition accuracy in real-time, minimising latency. With a fast response in real-time feedback, it will enhance user engagement, giving them constructive guidance for improvement.
Hand Position Visualisation	Computer vision plays an important role in this aspect of the software, ensuring that the hand remains consistently highlighted when positioned in front of the camera. This could simply be highlighting the hand with a simple square box, or adding a "skeleton" like in fingerspelling.xyz. It employs image processing to detect the hand's shape and position, followed by feature detection and extraction, which are crucial for the subsequent recognition phase of the software. In addition, all my stakeholders are right handed, making it easier for me to develop the software, as I will not have to focus on making a separate recognition algorithm for the left hand.

User Interface	To create a user-friendly GUI like fingerspelling.xyz, I'll rely on abstraction. This involves removing unnecessary features and avoiding excessive complexity. The interface should maintain a clean and straightforward appearance, ensuring intuitive use for our users. Recognition results will be displayed clearly.
Interactive Learning Environments	My stakeholders want this software to be utilised for developing their fingerspelling skills. The interactive exercise prompts users to fingerspell the first letter of displayed words, accompanied by real-time feedback on accuracy. Then the software will move onto the second letter of the word. This will then continue until the user reaches the end of the word, and a new word is shown.

System Requirements

Most contemporary computers should be able to run the software, but here are some specific requirements.

Content	Explanation
Operating System: Windows 10 or later	This software will be developed for Windows 11, ensuring compatibility with Windows 10. These are two of the most widely used operating systems, ensuring accessibility for a broad range of computer users.
Processor: Intel Core i3 (Recommended: Intel Core i5)	This software will be developed for Intel Core i3 processors, ensuring efficient performance on a wide range of computers, including those without high-end processors.
RAM: 4GB or more	This ensures your software runs smoothly and efficiently, with sufficient memory for processes like fingerspelling recognition, enabling real-time feedback and a seamless user experience. It also supports dataset storage for testing and training the model.
Camera: 720p or more HD webcam	It ensures the software captures fine hand movement details for precise recognition. Will help the model predict the sign with accuracy during the real time recognition.
Internet Connection	Fingerspelling recognition software relies on real-time processing and feedback. A fast internet connection reduces latency in data transmission between the user's device and the server.

Limitations

As developing a fingerspelling recognition software is complex on its own, adding these extra features will make the development of the software more challenging.

Desired Feature	Limitation
Gesture Recognition for Full Sign Language: extending recognition to full sign language beyond fingerspelling	Stakeholders may want full sign language gesture recognition for a more comprehensive learning experience, However, recognising full sign language is complex due to: - Extensive vocabulary, diverse gestures, and reliance on features like facial expressions and body language, which can introduce ambiguity. - Compiling a dataset for full sign language recognition is challenging, as it requires a diverse set of signs and complex data annotation. This extensive processing of visual data demands substantial computational power and memory.

Customisable Sign Language Variants: allowing users to choose their preferred sign language variant or regional dialect	Despite some of my stakeholder not having any knowledge on ASL, I chose to focus on this instead of any other fingerspelling variations due to following factors: - ASL involves simpler hand motions than some other sign languages. Supporting multiple sign languages with different complexities would demand much more data and computational resources, significantly complicating the development process. - Gathering enough training data for less common fingerspelling sign languages can be challenging, while ASL has loads of freely accessible datasets available.
Seamless Real-time Translation: offering real-time translation of sign language in diverse settings (e.g., during conferences or public events)	This feature would greatly help my stakeholders for learning fingerspelling and maximise user experience. However due to some limitations, this would not be fully achievable due to: - Background noise can interfere with the accuracy of hand gesture recognition. The software needs to distinguish the intended signing from the surrounding noise. Filtering out irrelevant sounds and focusing on the signer's gestures is a technical challenge. - Low light or harsh backlighting can prevent accurate hand gesture recognition. Adapting to various lighting conditions in real-time poses a technical challenge.
Integration with Other Software: integrating the software seamlessly with popular video conferencing or communication applications	One of my stakeholders (Robert) suggested this feature to be implemented in my software. However, this is limited by the following factors: - Developing a mod for an existing platform like Zoom requires extensive collaboration and access to the platform's APIs. Zoom's infrastructure might not readily support such modifications. - Translating video data into text may raise privacy concerns if not handled securely, as video calls often contain sensitive information.

I forwarded these features and system requirements to my stakeholders and posed the following question to confirm if they are satisfied with this final proposed solution for my software:

Stakeholder Feedback	
"Do you agree with these final proposed solutions, essential features, limitations and the minimum system requirement?"	
Yusuf	*"Yes. I don't mind the choice of sign language."*
Robert	*"Seems fine to me, good luck!"*
Imtiaz	*"Looks good, I would have preferred if you could have included the learning features in the essential features section, but if you think it is too much for you then leave it. It is important that you make the recognition part first. Overall I am happy with these features."* *"Thank you for your feedback. I will try my best to add something if I can."*

Success Criteria

Stakeholder feedback for the proposed solutions shows that they agree with it, therefore I can construct a full success criteria with all the information gathered in the Analysis section. This will be used as a guide to successfully develop the fingerspelling recognition software, ensuring it meets stakeholder expectations.

Requirement	Sub-Requirement	User Requirement	Justification	Success Criteria
Recognition	Camera access	Access user's front camera	This will generate input for my model, by capturing the user's hand position, which will be then sent to the model for recognition. This will also be used to make the bounding box of the hand. Camera access will facilitate real-time processing of fingerspelling, enabling instantaneous feedback to the user.	- Users should be able to see themselves from the front camera. - On the camera window, the right hand side will be covered with the GUI, and on the left hand side the user will be able to see his hand.
	Hand localisation	Bounding box	This will precisely define the region of the hand, isolating it and sending it to the model. It will enhance the accuracy of recognition, as the model will only require the image of the hand, ignoring the background. The bounding box will also serve as a visual cue for the user, indicating where the software is focusing its attention. This will also let the user know that the software is working in localising the position of the hand. This guidance is beneficial for new users who may be unfamiliar with the spatial requirements of gesture-based interaction.	- Box around the hand will be shown in every frame. - Box will follow hand movements. - Box will be x pixels away from the main joints of the hand. - One hand will be detected at a time, (software will not place a bounding box around another hand, if one hand is already being detected).
Real-Time Feedback	Output	Model accuracy	This accuracy is based on how well the model performs in recognising the sign. It provides users with feedback on the reliability of the recognition process, helping them measure the performance of the algorithm. Users can also understand where improvements need to be made in their fingerspelling. It acts as a form of constructive feedback, guiding users to refine their gestures, enhancing the user's learning experience. This is also important for me, as a developer, as I will be able to utilise this data to identify patterns of errors, update performance, ensuring software evolves to meet user's needs effectively.	- Display model accuracy as a percentage. - Display accuracy on the top left corner of the window. - Accuracy of the model will be medium sized.

		Letter prediction	Provides immediate feedback, allowing users to visually confirm whether the system has accurately recognised each letter in real-time. This instant feedback loop helps with the learning process by reinforcing correct gestures and helping users correct mistakes promptly. This will also enhance the engagement and user experience. Users can follow along the recognition output, actively participating in the learning process. This feature not only reinforces the learning of individual letters, but also helps users understand the composition of words in sign language.	- Display model prediction in the recognition of the sign (letter). - Display letter on top of the bounding box, on the left. - Letter will change based on the sign. - Letter should be a different colour than the bounding box.
Functionality	Interactive Learning Environment	Letter	It provides a structured learning path, allowing users to concentrate on mastering one letter at a time, which is crucial for building a solid foundation in fingerspelling. The software will first start with simpler signs, such as “A” or “B”, which will be part of the word to be fingerspelled. If the user signs the letter displayed correctly, and the recognition software correctly identifies the letter, the software will then move onto the next letter in the current word. This will continue until the user reaches the end of the word. The software’s capability to evaluate and provide instant feedback is important in allowing learners to recognise and improve their errors, enhancing the learning experience of the users.	- If the user's sign matches with the software's given letter, move onto the next letter. - Letter will be displayed in the centre of the right hand side of the window. - Size of the letter will be very big.
		Skip letter function	By giving users the option to skip a letter and proceed to the next one, the software accommodates varying learning paces and preferences, ensuring a more personalised and user-friendly experience. Secondly, this feature becomes particularly necessary if software fails to recognise certain signs. In such a case, the user needs to be able to move on, ensuring the learning process is not blocked by technical limitations. This problem may still occur, despite a model that recognises signs accurately. The software could utilise a simple method where each letter in a word is indexed by a string. By pressing the key to skip the letter, the algorithm would increase the index, effectively moving onto the next letter.	- Users should be able to skip to the next letter, with the press of a button (L). - Display message *“Press (L) to move onto next letter.”* underneath the letter. - Message will be underneath the current letter. - Message font size will be very small.

		Word	This feature is essential because it provides context and a clear objective for the user. This allows understanding how letters connect to form words, during fingerspelling. Additionally, seeing the whole word, encourages users to think ahead, preparing them for subsequent letters, which is beneficial for developing fluency in sign language.	- If the whole word is fingerspelled, move onto the next word on the list. - Word will be displayed underneath the letter to be fingerspelled. - Next word to be fingerspelled is also displayed next to the current word.
		Skip word function	This functionality will allow users to skip a word they find overly challenging or easy to fingerspell (due to the composition of the individual letters). Especially in learning environments, it's crucial to cater diverse skill levels and learning paces. Additionally, users will be able to see what word is going to come up, when they press the button to skip. By previewing the next word, users can make more informed decisions about whether to skip the current word, ensuring smoother transition while learning, therefore personalising their learning experience. The software could store all the words in a pre-existing list, and then display the word based on the index number on the list. As the user presses the button, I could just increment the index by 1, and display the next word.	- Users should be able to skip to the next word, with the press of a button (W). - Display next word, to let user know what word the user is going to fingerspell when he pressed the button. - Display message *"Press (W) to move onto next letter."* underneath the letter. - Message should be around a box. - Message will be underneath the current word. - Message font size will be very small.

	GUI colours	GUI background	Lighter colours tend to be easier on the eyes. The colour that I choose for the background of the GUI has a neutral and warm quality, making it appealing and non-distracting for a wide range of users. This is because I went for a colour scheme that will create a welcoming and comfortable interface, important for maintaining user engagement across diverse age groups.	- Background for the letters, words and instructions should be present on the left half of the screen.
				- Colour of the background should light:
				  - HEX: #FFE3B3
				  - RGB: rgb(255, 227, 179)

		Bounding box colour	The bounding box will be made out of a bright colour, which will ensure high contrast with the background, making it easily distinguishable. Bright colour draws attention to the hand region, guiding users to focus on the area where the recognition algorithm is active.
			
			This visual cue is especially beneficial for users who are learning and practising sign language, as it reinforces correct hand placement.
				- Colour of the bounding box should be a bright colour:
				  - HEX: #C957BC
				  - RGB: rgb(201, 87, 188)

		Text colour	Choosing this deep purple hue for the letters, words and accuracy is a strategic decision, primarily for its contrast against the light background. This colour stands out distinctly, making the text more visible and legible. This will draw the user's attention to the content, which will be the focal point of the software. This eventually helps in maintaining user's focus on the learning task, which is essential for skill acquisition. Additionally, this combination of colours creates an aesthetically pleasing interface.	- Colour of the text should be dark:
				  - HEX: #752092
				  - RGB: rgb(117, 32, 146)

	Instructions	Inform user to begin by placing hand in front of camera	This message will serve as an immediate instruction to users, especially those who are new to the software, guiding them on the initial step required to start the learning process. By instructing users to place their hands in front of the camera, the software can initialise the recognition process. This message indicates the software's calibration phase, where it adjust to specific lighting conditions, hand size and other environmental factors	- Display message *"Place hand in front of camera to begin."* when the user starts the program. - Remove message as soon as the hand is detected. - Message will be displayed underneath the word to be fingerspelled. - Message will be displayed in red.
		Inform user hand is not localised	It will guide users, especially beginners, on how to interact with the software, setting clear expectations for the hand's positioning necessary for successful gesture recognition. This message will also ensure that the software only initiates the recognition algorithms, as soon as the hand is detected. This can be simply done using a if function, to check if there are any coordinates for the bounding box. This will only be displayed at the start of the program, because I am going to assume that the user will already be familiar with how to use the software, on how to initialise the software, at that point. Additionally, this feature acts as a form of real time feedback, informing users when the software is ready to proceed, which is especially valuable in educational and learning environments where clear instructions and feedback are key to effective learning.	- Display message *"Place hand in front of camera."* only if the model can't localise the hand. - Remove message as soon as the hand is detected. - Message will be displayed underneath the word to be fingerspelled. - Message will be displayed in red.
		Inform user to position hand on the left hand side of the screen	This decision is primarily driven by the need for spatial separation between the user's hand and the GUI (displayed on the right). Since the GUI will overlay the right hand side of the screen, the user won't be able to see their hand if it is in that region. Therefore the user needs to be informed whenever the	- Display message *"Move hand on the left hand side of the screen."* - Remove message, as soon as the whole bounding box is present on the left hand side of the screen.

			bounding box is covered by the GUI. This will allow users to simultaneously focus on their hand movements, the feedback and instructions on the GUI. As most people are right hand sided, they are more likely to raise their right hand for signing, and the program will output a mirror reflection of the user, but it will be reflected vertically.	- Message will be displayed underneath the word to be fingerspelled. - Message will be displayed in red.
		Inform user correct hand spelling	Positive reinforcements through messages enhance the learning experience by providing immediate feedback. This positive feedback can overall boost user confidence, motivation and overall engagement with the learning process. Users are more likely to remain engaged and committed to learn sign language when they receive positive affirmations for their efforts.This may help users self-assess and reinforce muscle memory, in their fingerspelling.	- Display message *"Well Done!"* or *"Correct!"* if the user's sign matches with letter to be fingerspelled. - Message will be displayed underneath the word to be fingerspelled. - Message will be displayed in green.
	Stop Software	X button	This is important for user control and system efficiency. Users may need to exit the program for various reasons. This feature will provide a straightforward and immediate way for users to terminate the software. It will contribute to user satisfaction, allowing them to efficiently manage the application. User control aligns with user expectations, and improves the overall usability of my fingerspelling recognition software.	- The X button will be in the top left corner of the window. - When the mouse hovers over the X button, it will turn red. - If pressed, the window will close.

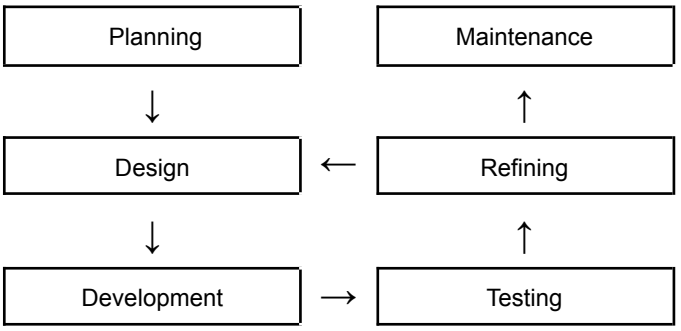
Design

This section will ensure that my project meets the expectations of stakeholders and fulfils the success criteria.

SDLC: Agile

The choice of a software development lifecycle depends on factors such as the project's size, complexity, requirements and structure.

Considering the characteristics of a fingerspelling recognition software, an agile development methodology would be appropriate. The use of this SDLC can be justified for the following reasons:



- **Adaptability to Changing Requirements:** requirements might change during the development, agile will allow me to be flexible with planning, allowing me to implement any changes late in the development process.
- **Incremental Development:** Developing an entire fingerspelling recognition software at once is very risky. Agile breaks down the development into small manageable iteration, where continuous refinement will help me to achieve optimal accuracy and user satisfaction.
- **User-Centred Approach:** The development of this software depends on user experience. Agile allows collaboration with stakeholders. Regular feedback during short development cycles ensures that the software aligns with user expectations and needs.

Problem Decomposition

The main objective of this project is to develop a real-time fingerspelling recognition system using object detection and train it with a dataset created using a webcam. The accuracy can be easily demonstrated through the model's predictions in detecting specific signs.

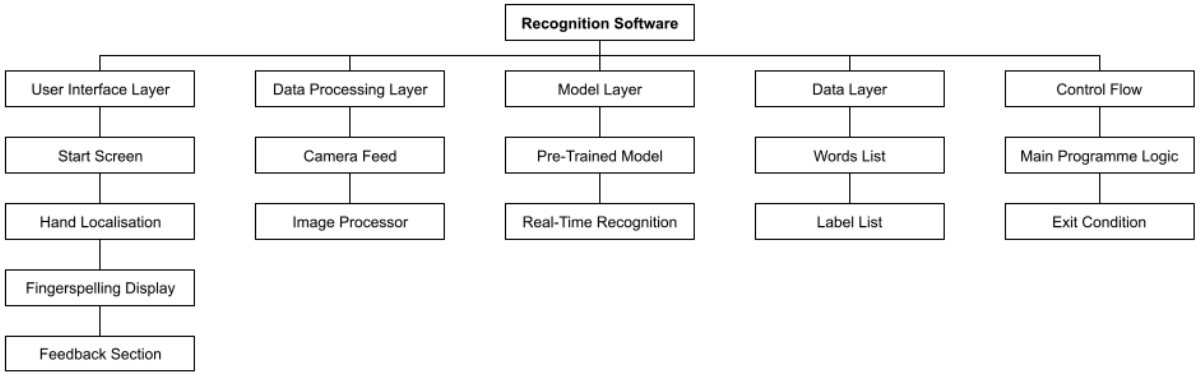
The development of this software will be split into two sections, to make it easier for me to work with:

1. **Development of Recognition Software:** This will be the main focus, as the development of these algorithms will require a vast amount of research. Machine learning differs from traditional programming, as the program is constructed based on supplied data and desired output, rather than expecting a predefined output. The key characteristic of machine learning algorithms is their ability to derive rules from existing examples [9]. This aspect will be explored further during the algorithm development process.
2. **Development of Learning Environment & GUI:** The GUI will feature a simple and lightweight design, ensuring ease of navigation. A clear menu with straightforward instructions will guide users through the software. Buttons for starting and stopping recognition will be incorporated, enhancing user control. The initiation of recognition will include a brief delay to load the hand tracking algorithm. The GUI will also display visual cues, such as highlighting the detected hand, to keep users informed about the system's status. For the learning part of the software, the user should be able to practise fingerspelling through given words. This will consist in spelling the words, through the fingerspelling recognition software.

This structured approach allows for a focused and systematic development process, addressing both the underlying recognition capabilities and the user interface to create a comprehensive and effective fingerspelling recognition software. I will mostly be seeking feedback from my stakeholders during the development of the GUI, as they do not need to know how the algorithm works. This approach ensures that the interface can be tailored to their specific needs.

Structure Diagram

The structure diagram of this software will be relatively simple, as all the focus will be on developing the recognition model, as it is complex.



1. **User Interface (UI) Layer:** This layer is engineered for intuitive navigation and user engagement, incorporating interactive elements that enhance the learning experience. It also adapts dynamically to user interactions, ensuring a personalised and responsive interface that caters to individual learning paces and styles.

2. **Data Processing Layer:** This layer is designed to optimise the quality and efficiency of image data processing, ensuring that the input is ideally suited for accurate gesture recognition. It also plays a pivotal role in adapting to varying lighting conditions and user environments, maintaining the consistency and reliability of the input data.

3. **Model Layer:** This core component is also responsible for dynamically adjusting the model's parameters based on user interaction, enabling personalised learning experiences. Furthermore, it continuously updates and refines the recognition algorithms as more user data is gathered, enhancing the system's accuracy and responsiveness over time.

4. **Data Layer:** This layer is also crucial for maintaining data integrity and ensuring consistent updates, which are vital for the evolving nature of the learning content. Additionally, it facilitates scalability, allowing for the easy addition of new words and labels to enrich the software's educational capabilities.

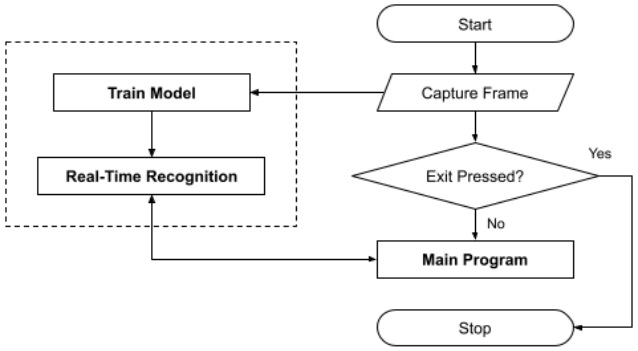
5. **Control Workflow:** This component, pivotal in managing the operational logic of the software, not only includes the main program logic that directs the application flow but also integrates the exit condition, allowing users to gracefully terminate the program. Additionally, it is equipped with error handling capabilities to manage exceptions and maintain the integrity of the user experience, ensuring the software remains stable and reliable under various conditions.

Together, these components are intended to provide a smooth user experience, from starting the software to engaging in interactive fingerspelling learning and ultimately exiting the program's window. The UI layer acts as the interface between the system and the user, providing information and gathering input from the user, while the data processing and model layers handle the complex tasks of image processing and gesture detection. The control flow maintains the overall program structure, ensuring logical progression and system stability.

Algorithms

Flow Chart - Overview of System Workflow

The flowchart below provides an overview of the framework of the fingerspelling recognition software.



The flow chart shows the main program for my fingerspelling recognition software, which illustrates a streamlined process that begins with the start of the program and proceeds with the real-time capture of frames.

Inside the dotted box, is where I will be developing the model, involving feeding labelled images, enabling it to learn specific patterns and shapes associated with the hand gestures. The iterative training process continues until the model reaches a satisfactory level of accuracy.

Once the model is trained, it interfaces with the 'Real-Time Recognition' component, where the live frames captured from the user's camera are fed into the model for prediction. The model processes these frames to predict and interpret the user's fingerspelling gestures in real time. The predictions are then sent back to the main program, which uses this information to verify if the user's gestures match the fingerspelling it is prompting the user to replicate.

The main program loop also checks for an exit command. If the user decides to stop, the 'Exit Pressed?' The decision point directs the flow to end the program. If not, the program continues to capture frames and interact with the recognition model.

Flow Chart - Development of Recognition Model

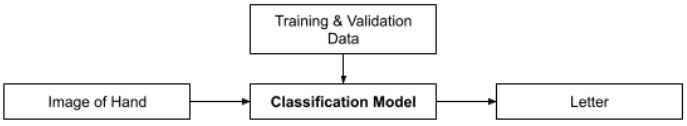
The ability for the computer to recognise a sign, is also known as object recognition. This consists of a series of tasks that the computer has to perform in order to recognise objects in a digital frame or video. This is split into 3 computer vision tasks:

1. **Object Localisation:** Locate presence of an object in the image.
   Input: Video frame of user fingerspelling.
   Output: If object is detected, outline object with a bounding box.

2. **Object Classification:** Predict the type of object in the image.
   Input: Image of the hand inside the bounding box.
   Output: Class Label (e.g. “A”, “B”, “C”, etc.) [10].

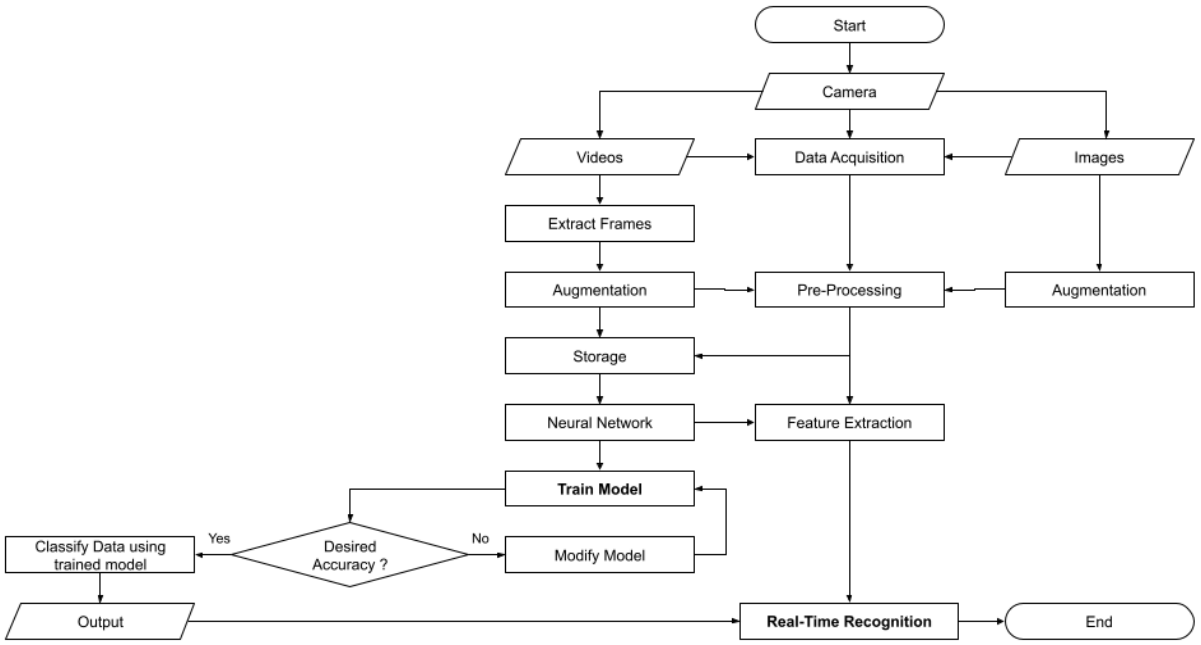
Therefore the final output when given a video frame, should be a bounding box around the object and the label, alongside with the percentage, indicating the accuracy of the prediction of the model, in classifying the sign.

Classification: a supervised machine learning method where the model tries to predict the correct label for the given data. Algorithms produce a list of object categories present in the image, which in this case, will be the frame captured from the camera, so that recognition is real-time.



On the right, the diagram shows how classification is essential for object recognition. Model is trained, using some training data, and then evaluated using some test data. Below is how the model is constructed.

Training the Model (Classification): The algorithm shows all the steps for training a model, in order to classify any object. This all depends on the data that it is given to the model.



1. **Data Collection:** By collecting images with my own hand for the data collection phase of my fingerspelling recognition software, I'm compiling a dataset that is specifically suited to the project. High levels of constancy in lighting, background, and hand placement will be guaranteed in this dataset, which are essential for the neural network to successfully identify and pick up the patterns required for precise fingerspelling recognition. This method enables quick iterations; if the model is inaccurate in identifying any gestures, I can quickly produce further pictures that concentrate on these areas, which will improve the model's accuracy. Furthermore, by using my own images, I sidestep the complexities and ethical issues associated with using third-party data, such as privacy and consent concerns.

2. **Data Preparation:** applying a skeleton (highlight main joints in the hand) to the images of my hand gestures significantly facilitates the training of the model. By focusing on the skeletal structure, the model can more effectively learn the key points that define each sign, leading to more precise recognition during use. Most importantly, ensuring that all images are of the same size is a critical step in data preparation. Consistent image size is essential because neural networks require fixed-size inputs. If the input dimensions vary, the network cannot apply the learnt filters and weights uniformly across the images.

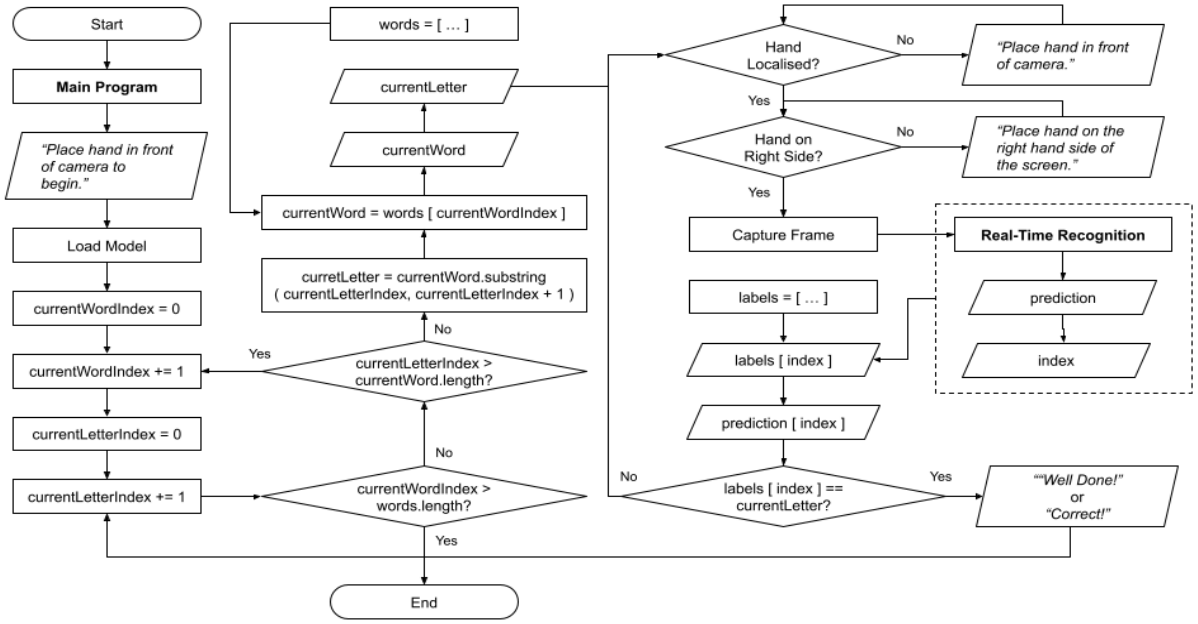
3. **Training Model:** After I will split the dataset into training, validation and testing

4. **Testing Model:** After looking at the results, of how accurately the model recognises each sign, I will be able to modify the training process by changing some hyperparameters in order to get the best accuracy possible. I can adjust settings, like the number of training epochs or the learning rate, to improve the model's performance, based on the initial accuracy during recognition. Another way to increase model's accuracy is to alter the dataset provided: adding more images and making sure images are different and not all the same (perhaps due to background).

Once the training is complete, the model can classify new images by running them through the same feature extraction process and using the trained classifier to output predictions.

Flow Chart - Development of Learning Environment

This flowchart below shows the operational logic of the learning environment within the fingerspelling recognition software. It serves as a roadmap that clearly illustrates the step-by-step process through which the software interacts with the user, processes input, and provides feedback.



1. **Initialisation:** This is where the program starts. It includes setting up initial variables and loading the machine learning model that will be used for recognising the fingerspelling.

2. **Word and Letter Management:** In this component, the software handles the progression through the list of words and their individual letters. It involves:
   - Selecting the current word from a predefined list (`currentWord`).
   - Iterating through the letters of the current word (`currentLetter`).
   - Checking conditions to progress to the next word or letter.

3. **User Interaction:** This part of the flow involves direct interaction with the user, providing instructions, and capturing the user's hand gestures via the camera feed. It ensures that the user's hand is localised and in the correct position before proceeding.

4. **Gesture Recognition Loop:** Once the user's hand is in position, the software captures a frame and processes it through the real-time recognition model, which predicts the fingerspelled letter.

5. **Feedback and Progression:** The software compares the predicted letter from the recognition model with the expected letter. If they match, it provides positive feedback to the user and progresses to the next letter or word.

6. **Program Termination:** The flowchart includes a decision point where the software checks if the user has indicated they want to exit, or there are no more words to iterate through the list. If so, the software will terminate, otherwise, it will continue with the next capture frame.

The two flowcharts provide an overview of the fingerspelling recognition software. The first flowchart outlines the model's development, highlighting the steps involved in preparing the model to recognise fingerspelling. The second flowchart illustrates the application's runtime behaviour, where the user interacts with the system to learn fingerspelling. Together, these charts illustrate a full cycle from model training to practical application, ensuring the system's effectiveness in teaching fingerspelling.

Dataset Directory Structure

As I will first develop the recognition function, I will need to collect images to feed into the model. The dataset will follow the train-validation-test split, as I don't want my model to over-learn from training data, and perform poorly after being developed. This split allows for the model to be trained on one set, tuned on another, and evaluated on a final set [13]. The split will consist in:

- **Training set:** 80%
  During this stage, my model will pick up information from pictures of hands making different ASL letters.Here, the majority of my collected images will be utilised, since repeated exposure to these examples is crucial for teaching the model to recognise the distinct patterns of each sign.
- **Validation set:** 10%
  This subset of the data is used to evaluate the model during the training process. It helps in tuning the model's hyperparameters and provides a check against overfitting.
- **Test set:** 10%
  The model will be tested with this set. If it performs well, it has successfully learned to generalise from the training data to new inputs.

Here is the structure of how the data is going to be split into different directories.

```
Data/
│
├── train/
│   ├── A/
│   ├── B/
│   ├── C/
│   ...
│   ├── X/
│   └── Y/
│
├── validation/
│   ├── A/
│   ├── B/
│   ├── C/
│   ...
│   ├── X/
│   └── Y/
│
└── test/
    ├── A/
    ├── B/
    ├── C/
    ...
    ├── X/
    └── Y/
```

This mimics how the model will be used in the real world and ensures its reliability and robustness when deployed for actual use.

Pseudocode - Image Dataset Sorting

The main reason I choose to write the process of sorting the dataset in pseudocode is because it helps me to abstract away from the syntax-specific details of programming languages, enabling me to concentrate on the algorithm's logic and structure. This abstraction is particularly beneficial given the complexity of handling file operations and the necessity of ensuring a balanced dataset split.

```
// Import necessary operations for file manipulation (not explicitly shown in
pseudocode)

// Define main directory and subdirectories
SET mainDirectory TO "Data/"
SET subDirectories TO ["train", "validation", "test"]
SET letters TO ["A", "B", ..., "Y"] // List all relevant letters

// Ensure each letter has a subdirectory in train, validation, and test
FOR EACH subDirectory IN subDirectories
    FOR EACH letter IN letters
        CREATE directory mainDirectory + subDirectory + letter IF NOT EXISTS

// Define function to split and distribute images
DEFINE FUNCTION split_data(letter, images)
    CALCULATE splits for training, validation, and testing
    DISTRIBUTE images into respective directories based on splits

// Iterate over each letter, process images, and organise into new structure
FOR EACH letter IN letters
    SET letterPath TO mainDirectory + letter
    IF letterPath exists THEN
        GET all images in letterPath
        CALL split_data WITH letter and images
        REMOVE original letter directory
```

1. Setting Up the Workspace: The code first defines where all the data will be organised (mainDirectory) and lists the categories (letters) and types of data sets (subDirectories) it will work with.

2. Creating Necessary Folders: For each type of dataset (train, validation, test), it creates a specific folder for every category letter if those folders don't already exist. This ensures there's a place to put each image based on its category and which dataset it belongs to.

3. Dividing Images into Datasets: The code takes all images from an original folder (one for each letter) and divides them into three groups: 80% of the images go to the training set, 10% to the validation set, and the remaining 10% to the test set. This split helps in machine learning tasks where the model is trained on the training set, fine-tuned with the validation set, and finally evaluated on the test set.

4. Moving Images: After dividing the images, the code moves them to their respective folders within the "train," "validation," and "test" directories based on the image's category.

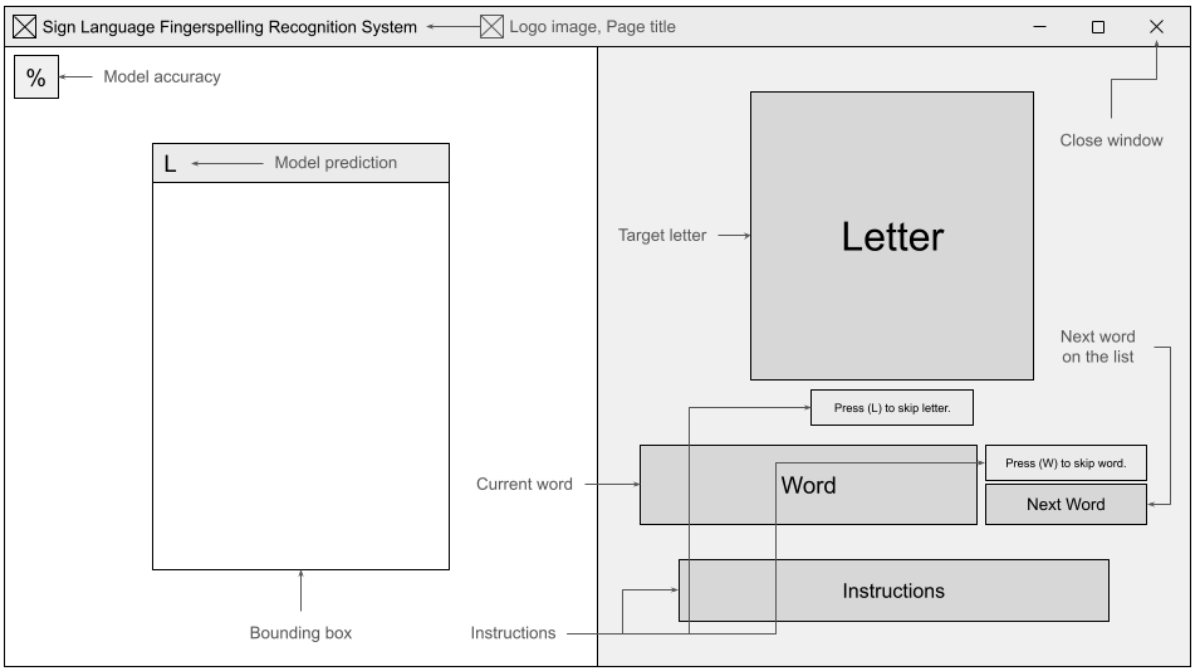
5. Cleaning Up: Once all images have been moved to their new locations, the original folders (which are now empty) are deleted to tidy up the workspace.

Since I've written my solution in pseudocode, stakeholders who might not be familiar with programming can easily understand my approach to this issue.

GUI Design

The GUI is the interface that connects users to the software and is intended to be easy, accessible, and efficient. My key goal in creating the GUI is to ensure that it not only allows for easy navigation but also improves the learning experience for users.

The suggested GUI shown below has similarities to the existing solutions that I found during my investigation.



The layout of the GUI was thoughtfully crafted to be straightforward and user-friendly. The GUI is intuitive, with self-explanatory controls and a logical layout that users can easily navigate from the outset. I believe that a clear interface is crucial, especially for educational tools, as it allows users to focus on learning without being overwhelmed by complex navigation or excessive features.

Usability Features

Each of these qualities contributes to the software's usability, accessibility, and efficiency, resulting in a better overall experience for the user.

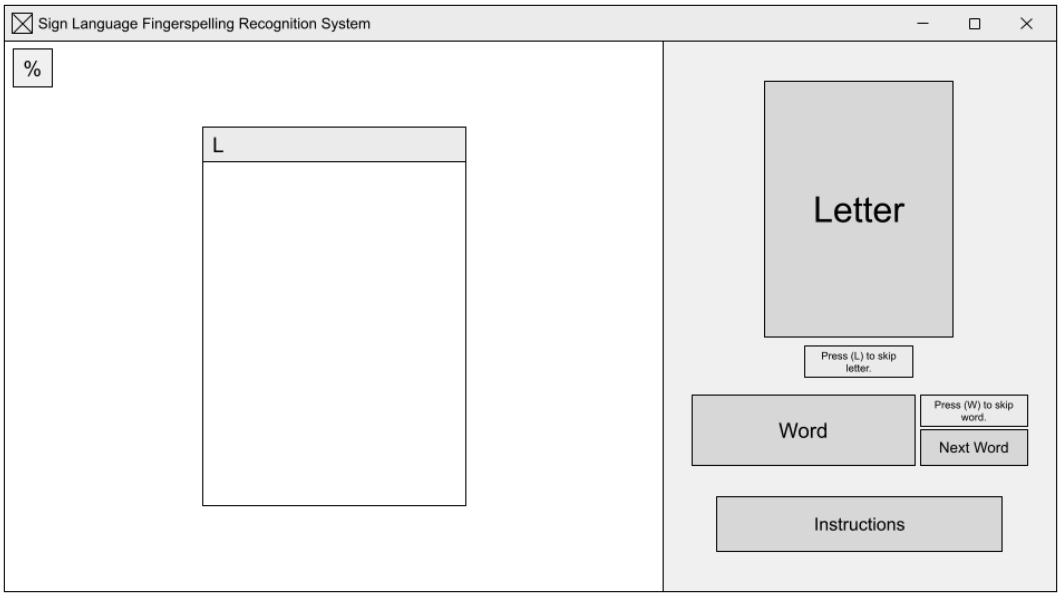
Colour Scheme: The background is a soft beige (HEX: #FFE3B3) , offering a calm setting, while letters and words pop in deep purple (HEX: #752092) for clear visibility. Instructional text in lighter purple (HEX: #C957BC) distinguishes itself from other texts. This scheme not only appeals visually but also aids in easy distinction of key elements, supporting user focus and understanding.

- **Bounding Box:** The bounding box, vital for hand gesture recognition, is brightly coloured in HEX: #C957BC for high visibility. This distinct hue prevents blending with the background, helping users easily locate where to position their hands for precise gesture detection.
- **Responsive Feedback:** The software will offer instant colour-coded feedback for user actions. For example, correctly signing a letter prompts a green "Well Done" message. This feature effectively indicates user performance and encourages progress.

Font and Size: I have chosen a font that is easy to read (`HERSHEY_SIMPLEX`), with an appropriate size and that ensures all users, regardless of their visual abilities, can comfortably read the text without strain.

Logo: While the logo is a less critical aspect of usability, it still contributes to the overall aesthetic. However, our main focus remains on the functional aspects that directly impact user experience.

I also presented this GUI design and usability features to my stakeholders. Their perspectives were invaluable in ensuring that the design was not only technically sound but also met the practical needs of our users.

Stakeholders Feedback	
"Is there anything about this interface that you would like to change, or are you satisfied with the current design and functionality?"	
Yusuf	*"I really like the design; it's clean and focused. However, it would be great if there was more space to view the hand clearly during signing. That would help me make sure I'm doing the signs correctly."* *"Thank you for the feedback. I'll make the adjustments to give a larger view for the hand recognition area and will share the updated design with you."*  *"This looks much better! Thanks for making the change."*
Robert	*"The design of the software is less of a concern to me; the functionality and its ability to enhance my communication skills in the workplace are what truly matter. I look forward to integrating it into my daily practice."*
Imtiaz	*"I'm really pleased that you decided to include the feature that shows the next word. It will help me prepare in advance and make the learning process smoother. Can't wait to get started and see how it helps me improve my sign language skills."* *"Based on the feedback I've received, I have made some improvements to the software, particularly in enhancing the hand recognition area for better visibility. I'm excited to show you this updated version and would love to hear your thoughts on it"* (show improved version of GUI). *"This new version is even better! The larger view for hand recognition is a great improvement."*
"Do these usability features meet your expectations for an effective and user-friendly learning tool?"	
Yusuf	*"Yes, I like the idea of the bounding box being bright. Very smart."*
Robert	*"Yes, sounds fine."*
Imtiaz	*"Sure, the usability features look great and are well thought out. However, can I suggest using a simple emoji for your logo?"* *"Thank you for your feedback. I will take that into consideration."*

Key Variables

Here are the key variables that play an important role in the functionality and logic of the software.

Section	Method	Name	Data Type	Justification
Camera	Variable	`cap`	Object	This VideoCapture object is essential for initialising the video stream from the webcam, allowing the software to capture live video for hand detection and gesture recognition. It directly impacts my stakeholder's ability to interact with the software in real-time.
	Variable	`detector`	Object	The HandDetector object is crucial for identifying and tracking my stakeholder's hand within the video feed. It enables the core functionality of recognising fingerspelled letters, meeting the educational goals of the software.
	Variable	`img`	Object	This variable represents the current frame captured from the video stream. It is essential for detecting hand gestures in real-time, serving as the primary input for both hand detection and subsequent gesture recognition processes.
	Variable	`x, y, w, h`	Integer	These variables represent the coordinates and dimensions of the bounding box around the detected hand (x and y for top-left corner, w and h for width and height). They are crucial for isolating the hand in the video frame, which is essential for accurate hand gesture recognition and for ensuring the cropped image encompasses the entire hand gesture for classification.
Classifier	Variable	`classifier`	Object	Classifier is key for loading the pre-trained machine learning model and labels for gesture classification. It supports the software's objective of teaching ASL by accurately recognising and interpreting fingerspelled letters.
	Function	`get_prediction`	Function	Retrieves the model's prediction for the currently detected hand gesture. This functionality is at the heart of the software's educational feedback loop.
	Array	`modelAccuracy`	Float	This variable captures the confidence level of the classification model's predictions of individual letters. It's crucial for providing my stakeholders with feedback about the accuracy of their fingerspelling.

	Variable	`index`	Integer	Identifies the highest confidence score's label from the prediction array, determining the most likely gesture made by the user. This is critical for mapping the model's output to a specific fingerspelled letter, directly influencing the feedback provided to my stakeholders and facilitating the learning process.
	Array	`labels`	String	Contains the set of letters that the model can recognise. It's directly linked to the learning outcomes, allowing my stakeholders to engage with and learn the entire ASL alphabet effectively.
	Variable	`letter`	String	Temporarily stores the letter predicted by the model, facilitating comparison with the actual letter the user is attempting to fingerspell. This comparison is central to providing immediate, actionable feedback to my stakeholders regarding their performance.
Learning Environment	Variable	`currentLetterIndex`	Integer	Manages the position within the current word, ensuring my stakeholders can practise fingerspelling word by word, letter by letter.
	Variable	`currentLetter`	String	Represents the specific letter within a word that the user is currently focusing on.
	Function	`words_list`	Function	Reads the practise words from a file into a list, making a variety of practise words available to my stakeholders.
	Directory	`words.txt`	Text File	Serves as the external data source from which the words list variable is populated. Containing a curated selection of words suitable for ASL fingerspelling practice, this file directly influences the breadth and diversity of the learning content available to the user.
	Array	`words`	String	Acts as the repository for the collection of words that users will learn to fingerspell. This list is derived from the words.txt file, ensuring a diverse and comprehensive set of vocabulary for practice.
	Variable	`currentWordIndex`	Integer	Tracks the current position in the list of words, facilitating structured learning progression through the word list.
	Array	`currentWord`	String	The current word the user is attempting to fingerspell, broken down into individual letters.
Buttons	Variable	`key`	Integer	Captures user input from the keyboard to navigate through the software, such as skipping letters or words. This functionality will enhance my stakeholder interaction, allowing for a more tailored and engaged learning experience.

Test Data for Iterative Development

This table will allow me to continuously refine and enhance the software's capabilities through repeated cycles of testing and modification.

Test				Expected Result
Type		Input	Justification	
Data Collection	Normal	Webcam access	Testing webcam access ensures the software can initiate the primary mode of input, as stakeholders require reliability in capturing user gestures for the educational purpose of the software.	Display webcam feed.
	Normal	Hand detected in frame	It's crucial to test hand detection to verify the software's capability to identify and track the user's hand, as stakeholders prioritise accurate recognition for effective ASL learning.	Display annotated hand and bounding box on webcam feed.
	Invalid	No hands in frame	This test checks the software's stability and error-handling when expected input is absent, which is critical for stakeholders to guarantee a non-disruptive user experience.	Do not crash or freeze, continue displaying webcam feed.
	Normal	Hand images are different size	Testing with various hand sizes ensures the system is flexible and inclusive, meeting stakeholder objectives to accommodate users with different physical characteristics. Also because the machine learning model requires fixed size input, I will require it to make input constant size.	Resize image to square, maintaining proportionality.
	Normal	Press 's' to save image	Testing the 'save image' function is to ensure that the data collection process is efficient, enabling me to quickly and easily create a dataset for training the model.	Save image to folder and print letter, counter.
	Normal	Save image	Verifying that images are saved correctly is vital for maintaining the integrity of the training dataset. Accurate and consistent image capture is necessary to train a reliable and effective model.	Save the same amount of images in each respective folder.
Sorting Dataset	Normal	Data/ ├── A/ ├── B/ ... ├── X/ └── Y/	Structuring the dataset into training, validation, and test splits is essential for developing a robust machine learning model. The training set is used to teach the model to recognise patterns, the validation set to fine-tune the parameters and prevent overfitting, and the test set to evaluate the model's performance on unseen	Data/ ├── train/ │ ├── A/ │ ├── B/ │ ... │ ├── X/ │ └── Y/ ├── train/ │ ├── A/

			data. This separation is critical for ensuring that the model generalises well and performs reliably in real-world scenarios.	`│ ├── B/`
				`│ ...`
				`│ ├── X/`
				`│ └── Y/`
				`└── validation/`
				`├── A/`
				`├── B/`
				`...`
				`├── X/`
				`└── Y/`

	Boundary	Images in each subdirectory in `Data/`.	Allocating 80% of the data to training is essential for providing the model with a sufficiently large set of examples to learn from, improving its ability to recognise and generalise from the training data.	Move the **first** 80% of images from each letter directory in `Data/` to each letter in the `train/` directory.
	Boundary	Images in each subdirectory in `Data/`.	Using 10% of the data for validation is standard practice for tuning the model's hyperparameters and making decisions about the model without overfitting to the training set.	Move the **next** 10% of images from each letter directory in `Data/` to each letter in the `validation/` directory.
	Boundary	Images in each subdirectory in `Data/`.	The remaining 10% of the data is reserved for the test set to evaluate the model's performance. This set acts as a final, unbiased benchmark to assess how well the model will perform on new, unseen data.	Move the **last** 10% of images from each letter directory in `Data/` to each letter in the `test/` directory.
	Invalid	Letter directory does not exist	If a letter directory is missing, no action is taken as the dataset must represent all characters to be a valid training, validation, or testing set. This ensures the model's ability to recognise the entire alphabet.	Do nothing.
Model Development	Normal	Evaluate model accuracy.	Achieving over 95% accuracy ensures that users experience minimal frustration from misrecognition and can learn fingerspelling efficiently.	Test Accuracy > 95%.
	Normal	Save model as a h5 file	When using tensorflow or keras, saving the model as a h5 file is particularly effective for deep learning models, which can be quite large, as it efficiently compresses and stores not only the model architecture but also the weights and training configuration [11].	model.h5
Recognition Model Setup	Normal	Clear image of a hand sign.	Ensuring clear image capture is crucial for stakeholders, as it directly affects the model's ability to accurately classify hand signs.	Correct classification of hand sign.

	Normal	Hand detected in camera feed.	Reliable hand detection and visual feedback with bounding boxes are important to stakeholders to verify the system's responsiveness and accuracy in real-time usage.	Draw a bounding box around hand, with model prediction on top.
	Normal	Hand signs with different orientations.	Testing different orientations ensures the model can accurately classify hand signs from users regardless of their hand positioning.	Correct classification of hand sign.
	Boundary	Very low lighting conditions	Validating the model's performance in low light conditions ensures that the software is usable in various environments.	Correct classification of hand sign.
	Boundary	Hand sign at the edge of camera frame.	Assessing the model's response when a hand sign is partially out of frame is important to prevent false positives and maintain the integrity of the system, aligning with stakeholders' emphasis on accuracy.	Do not detect hand.
	Invalid	Completely dark frame	Ensuring the software does not crash or behave unexpectedly in the absence of visual data is crucial for a robust user experience, addressing stakeholders' expectations of system reliability.	Do nothing.
	Boundary	More than one hand detected in the same frame.	Ensuring the software can prioritise a single hand when multiple are present is necessary for focused learning sessions, in line with stakeholders' educational goals.	Continue sending image of first hand detected to the model.
	Invalid	Non-hand object in the frame.	It's necessary to prevent false detections of non-hand objects to avoid confusion and ensure a streamlined learning process, meeting stakeholder requirements for clarity and precision.	Do not detect an object as a hand.
	Normal	Hand sign with a slight blur motion.	It's necessary to prevent false detections of non-hand objects to avoid confusion and ensure a simple learning process, meeting stakeholder requirements for clarity and precision.	Correct classification of hand sign.
	Boundary	Fast-moving hand sign.	Testing the model's response to rapid movements is important to set practical usage boundaries and maintain classification accuracy, which stakeholders consider vital for the educational utility of the software.	Do not detect hand.
	Invalid	Extremely blurred or obscured hand sign.	Ensuring the software dismisses heavily blurred images prevents inaccurate classifications, supporting stakeholders' interest in a reliable recognition system.	Do not detect hand.

	Normal	Different skin tones.	The model must recognise hand signs across a spectrum of skin tones, ensuring inclusivity and broad usability, which is a key stakeholder concern.	Detect hand.
	Normal	Hand close to camera.	Close proximity ensures optimal visibility and detail for accurate hand sign detection, crucial for maintaining user engagement and learning efficacy.	Detect hand.
	Invalid	Hand very far away from camera.	Extreme distances may impair detail and accuracy, testing ensures the system doesn't misclassify or generate false positives, aligning with the requirement for precise recognition.	Do not detect hand.
	Normal	Hand sign for each letter (except J and Z).	Comprehensive recognition across the alphabet (with specific exceptions since they require movement of hand, instead of stationary sign) demonstrates the system's capacity to support users in learning a wide range of signs.	Correct classification of hand sign.
	Boundary	Any other hand gesture that is not part of the ASL alphabet.	Evaluating non-ASL gestures ensures the model's resilience against erroneous inputs and aligns with the goal of providing an accurate learning tool for ASL students.	Classify sign based on which hand gesture it closely resembles.
Learning Environment	Normal	Letter prediction given by the model matches with the letter of the word.	This is essential to ensure that the software responds correctly when users sign correctly, providing positive reinforcement which is crucial for effective learning.	Move onto next letter.
	Invalid	Letter prediction given by the model does not match with the letter of the word.	The system should not proceed when a sign is incorrect, as it encourages the user to try again, fostering learning through repetition and correction.	Do nothing.
	Boundary	Letter prediction given by the model does not match with the **last** letter of the word.	It's important to test the software's ability to handle the end of a word, ensuring that users have a seamless transition to the next learning step.	Move onto next word.
	Boundary	Word is last word on the list	The system should recognise the completion of the set words list and stop, providing a clear indication to the user that the session is complete, which is necessary for structuring the learning experience.	Stop programming.
Skip Letter/Word	Normal	Press (`L`) key.	Ensures that stakeholders can efficiently progress through the learning module without unnecessary delays. Also this is	Move onto next letter in the word.

			necessary as the model may not recognise the sign correctly, and therefore allows my stakeholders the control over the software, moving at their own pace.	
	Boundary	Press (`L`) key and `currentLetter` is the last letter of `currentWord`.	Ensures the software supports the natural flow of language learning, transitioning seamlessly between words.	Move onto the first letter in the next word.
	Normal	Press (`W`) key.	Stakeholders require the softwareX to advance through the learning material systematically, especially when a word is well-practised or correctly recognised.	Move onto the next word in the list of words.
	Boundary	Press `(W)` key and `currentWord` is the last word in `words`.	The app should enable continuous practice by cycling the word list, demonstrating its capability to provide uninterrupted learning sessions.	Move onto the first word in the list.
	Invalid	Press any other key.	It is essential that the application ignores irrelevant inputs to prevent disruption of the learning flow, ensuring a focused learning environment.	Do nothing.
Instructions	Invalid	No hands detected before model is loaded	Stakeholders need assurance that the software prompts the user to start interaction only when it's ready, preventing confusion and ensuring a smooth user experience.	"Place hand in front of camera to begin." message displayed.
	Normal	Hand position is out of predefined boundary	It’s important for the user to receive immediate feedback if their hand is outside the designated tracking area to ensure effective gesture recognition.	"Move hand on the left hand side of the screen." message displayed.
	Boundary	Hand position is at the edge of the camera view and GUI.	The stakeholders require the system to guide users in real-time, ensuring that the hand is within the capture frame for accurate recognition.	"Move hand on the left hand side of the screen." message displayed.
	Invalid	No hand detected.	The software must prompt the user to present their hand for detection, signifying robust error handling for non-ideal scenarios.	"Place hand in front of camera" message displayed.
	Normal	Correcte gesture for a letter.	Stakeholders want to ensure that correct recognition is positively reinforced, encouraging continuous use and learning.	Random congratulatory message ("Well Done!" or "Correct!") displayed.

Test Data for Post Development Testing

These requirements provide a way to evaluate the effectiveness, reliability, and performance of the software. I aim to provide an effective and user-focused solution that satisfies the wide range of needs of our user base by using this rigorous testing approach.

Action	Expected Result	Related Success Criteria
Initiate software	The webcam is activated, and video feed begins	Users should experience no delay or issues in the initialisation of the video capture for hand recognition.
One hand is present within the video feed	Hand is detected and a bounding box appears	Users should have their hand detected consistently for gesture recognition to take place.
Hand performs a sign from the ASL alphabet.	Correct letter from ASL is recognised	Users should receive immediate and accurate recognition of their hand gestures corresponding to the ASL alphabet.
Navigation through Letters and Words	Users can cycle through the ASL alphabet and words for practise	Users should be able to navigate through letters and words seamlessly for an uninterrupted learning experience.
Interaction with the GUI	User actions trigger appropriate responses within the interface	Users should find the GUI elements responsive and interactive, enhancing the learning process.
Model recognises the sign	Current prediction accuracy is displayed as a percentage	Users should be provided with real-time feedback on the accuracy of their gestures to facilitate self-assessment.
User does not know how to use software	Appropriate instructional messages are displayed depending on user interaction	Users should be guided by clear, contextual instructions to enhance understanding and usability.
Hand is detected, but it’s not in the predefined frame.	When hand is outside the defined frame, the system prompts the user to move their hand into view	Users should be alerted when their hand is not correctly positioned, ensuring they are always ready for gesture recognition.
Hand sign recognised by the model matches the target letter	The system advances to the next letter or word	Users should be able to proceed to the next letter or word upon successful recognition, enabling progressive learning.
Appropriate skip button is pressed	The software skips the current letter or word and advances as intended, if respective button is pressed	Users should be able to control their learning pace by skipping letters or words as needed.

Development

Creating the coded solution iteratively, using the plan outlined in the Design phase as a foundational guide.

Structure

The development process, if required, will stick to the Design section in order to make sure that no aspect is missed and to avoid major changes.

The majority of iterations will include testing, development, introduction, stakeholder reviews, and my own review. Before testing the complete iteration, each individual or group function and procedure will be written, tested, and explained as stand-alone entities to work out any bugs.

Technical Setup and Tools

Utilities used during the development process. If there are any changes, I will make sure to mention it during the development process.

Hardware Used

- **Operating System:** Windows 10

- **Hardware Environment:**
  - Processor: Intel(R) Core(TM) i3-1005G1 CPU @ 1.20GHz 1.19 GHz
  - Installed RAM: 8.00 GB
  - System Type: 64-bit operating system, x64-based processor

- **Camera:** 720p HD

Software Used

- **Interface:** PyCharm Edu (Version: 2023.3.3)
  As a free edition of PyCharm, it offers an affordable and accessible solution for the development of my project, removing cost as a potential barrier. It is also suitable for managing project dependencies, particularly the installation and management of external packages. Additionally, PyCharm Edu is equipped with various helpful tools and features, including code completion, debugging support, and version control integration, all contributing to a more streamlined, efficient, and error-free development experience.

- **Software:** Python (Version: 3.9.0)
  Python coding will be used for the project in order to simplify development. By using this version as it is supported with the version of TensorFlow that I will be using [12]. This choice ensures that I work in a stable and supported environment because TensorFlow releases are carefully tested with particular Python versions to ensure compatibility and best possible performance. Also it minimises the risk of encountering unexpected bugs and maximises the efficiency of my development process by ensuring that I am utilising a well-supported configuration.

- **Backups:** GIthub and Git
  I will use Git to upload and update my code on a regular basis in a private GitHub repository. This will make it easier to transfer the code between machines and give me a safe way to store backups, which will improve the project's accessibility and data security.

Packages Used

- **OpenCV:** (Version: 4.9.0.80) open-source library for computer vision, image processing and machine learning. It is primarily used for real-time applications, using images and video analysis, providing set functions such as image processing, object detection, feature extraction and more, which are essential for identifying and interpreting the gestures made during fingerspelling. This will be mainly used to access the device's camera and take the input from there.

- **CVZone:** (Version: 1.6.1) Another toolkit for computer vision that makes using AI in image processing tasks easier. The characteristics of CVZone, especially its hand tracking and identification abilities, are really helpful for our program. Fingerspelling interpretation requires accurate feature extraction and gesture classification, which we can accomplish quickly by utilising CVZone.

- **MediaPipe:** (Version: 0.10.9) A powerful cross-platform framework for building multimodal (video, audio, any time-series data) applied machine learning pipelines. Its ability to provide high-fidelity hand and finger tracking allows us to capture the nuances of sign language fingerspelling accurately.

- **Numpy:** (Version: 1.26.3) It provides support for multi-dimensional arrays and matrices, along with a large collection of mathematical functions to operate on these arrays. In our context, Numpy is particularly valuable for image data manipulation, as it allows for the efficient transformation of captured images into matrices, which can then be readily processed and analysed by our machine learning algorithms.

- **TensorFlow:** (Version: 2.9.1) An open-source machine learning framework that provides a comprehensive set of tools and libraries for building and deploying complex machine learning models. It will be used to train, evaluate, and deploy deep learning models capable of recognising and classifying hand gestures. Its ability to handle large datasets and perform efficient numerical computation accelerates the training process and enhances the model's accuracy. Also this version is compatible with the Python version that I am using.

- **Protobuf:** (Version: 3.19.6) Protocol Buffers (Protobuf) is a language-neutral, platform-neutral, extensible mechanism for serialising structured data, similar to XML, but smaller, faster, and simpler. In our software, Protobuf is used for defining the data schema and serialising structured data. This serialisation is crucial when saving the state of our machine learning models or when transmitting data between different components of the application. Protobuf ensures that the data structure is maintained and that the data exchange is efficient and compatible across various systems, which is vital for the reliability and scalability of our fingerspelling recognition software.

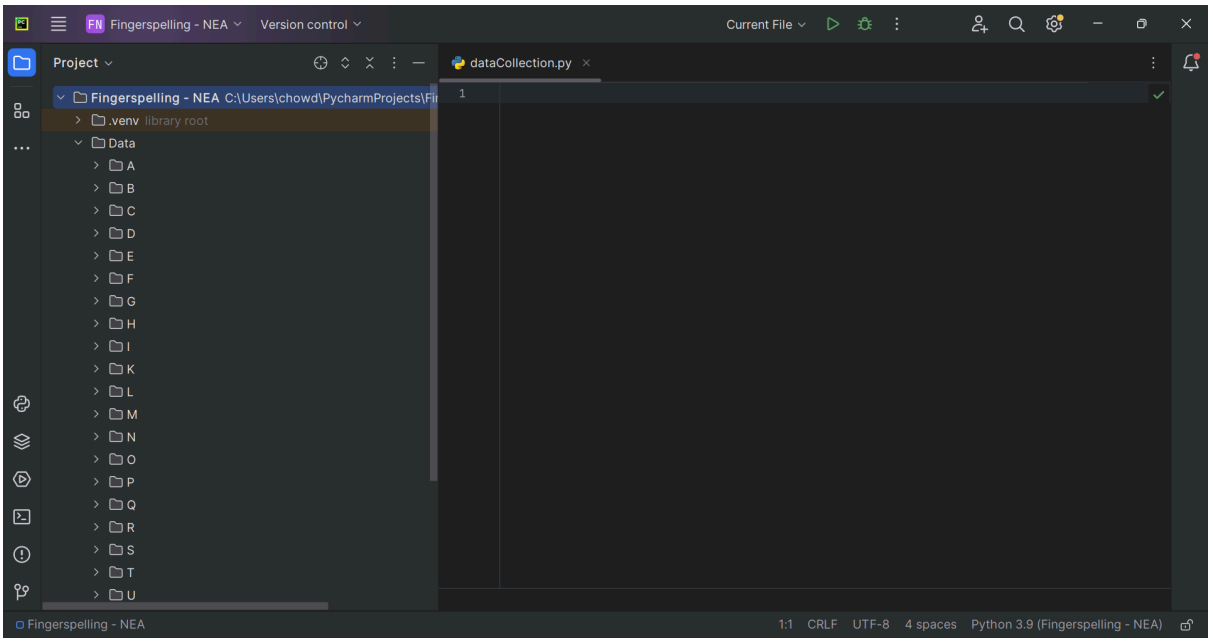
Other additional packages will be installed alongside these main packages. If I happen to change any of these packages, or need an additional library I will make sure to mention it during the development.

Developing Recognition Model

1st Iteration - Data Collection

Since this section primarily involves coding and collecting images for model preparation, it won't necessitate much stakeholder feedback.

To facilitate the collection of images, I have created a directory for each letter that the model will recognise, with each directory labelled with its corresponding letter. I will divide the data after I collect the images for each letter.



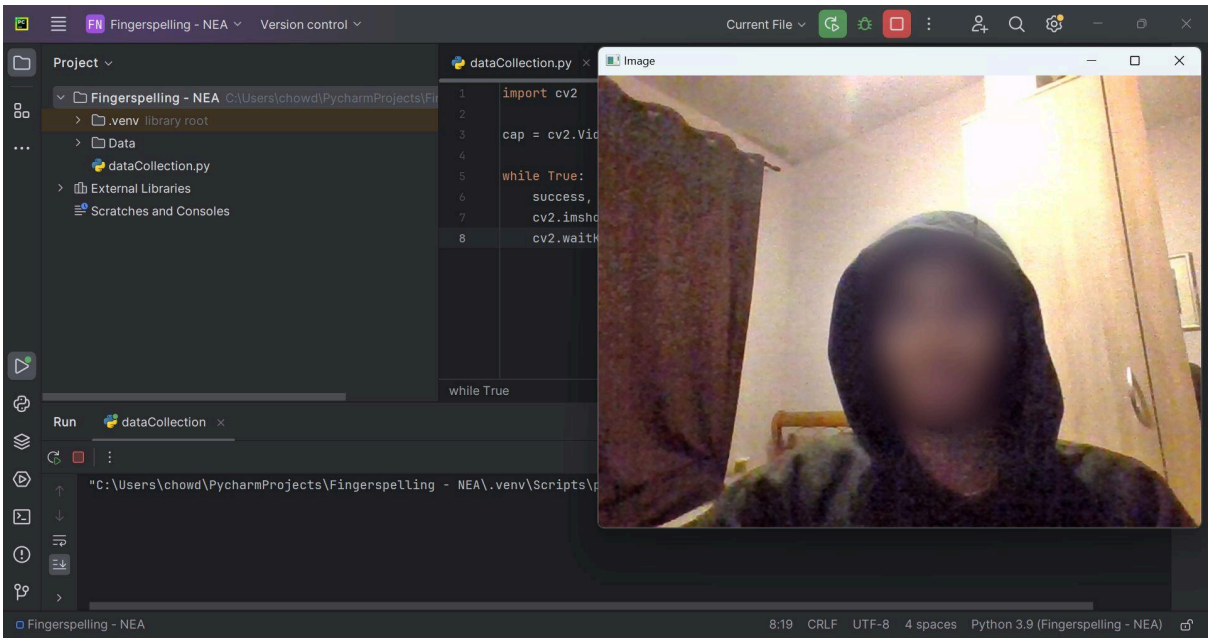
First thing that I will do is to make sure that the webcam can be accessed, and works properly. This is done simply using OpenCV.

```
import cv2

cap = cv2.VideoCapture(0)  # Initialise video capture

while True:
    success, img = cap.read()  # Capture frame from camera

    cv2.imshow("Image", img)

    cv2.waitKey(1)
```

The `cv2.VideoCapture(0)` is crucial as it initialises the video capture object for the camera. The index `0` is the default camera. Inside the loop, `cap.read()` continuously captures frames from the camera. `cv2.imshow()` is used to display each frame captured by the camera in real-time.

Finally, `cv2.waitKey(1)` waits for a key press for `1` millisecond and then continues to execute the loop. This is essential as the `while True:` loop with this code at the end ensures that the frame display is updated continuously.

Output:



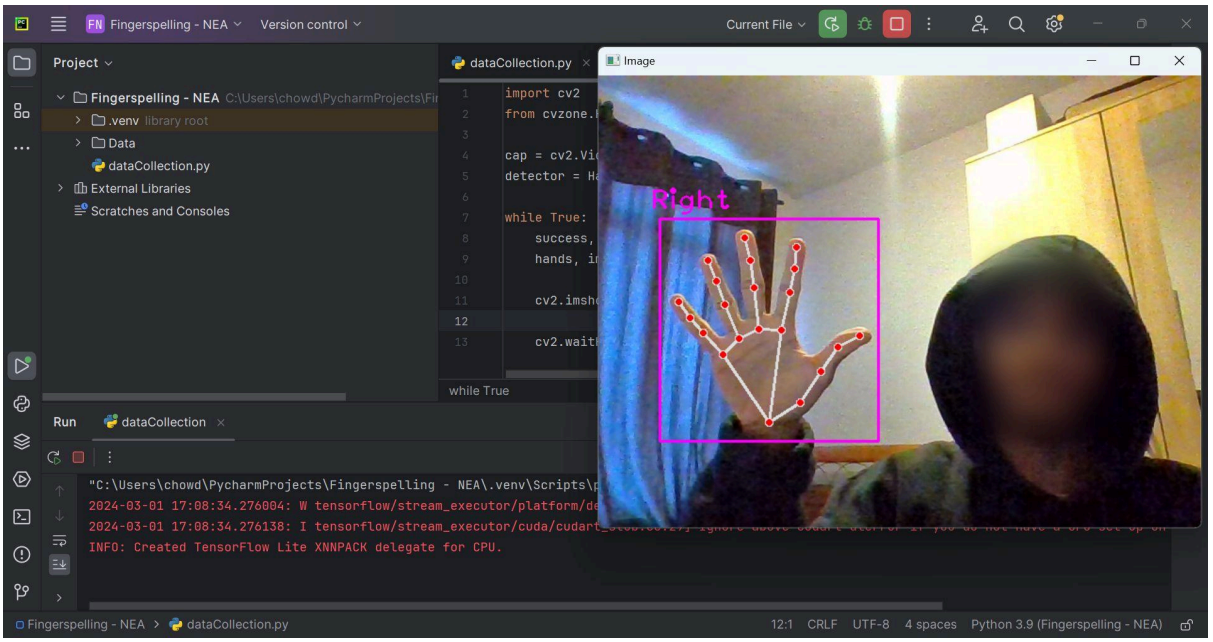
Now I will be importing the hand tracking module from CVZone.

```
import cv2
from cvzone.HandTrackingModule import HandDetector

cap = cv2.VideoCapture(0)  # Initialise video capture
detector = HandDetector(maxHands=1)  # Initialise HandDetector object

while True:
    success, img = cap.read()  # Capture frame from camera
    hands, img = detector.findHands(img)  # Detect hands and annotate the frame

    cv2.imshow("Image", img)

    cv2.waitKey(1)
```

An instance of the `HandDetector` class from the `cvzone.HandTrackingModule` is created and assigned to the variable `detector`. The `maxHands=1` argument specifies that the detector should track a maximum of one hand within the video frame at any time.

The `findHands` method of the `HandDetector` object (`detector`) is called with the current frame (`img`) as the input. This method processes the image to detect and locate hands within it. It returns two items: `hands`, which is a list containing the details of each detected hand (such as position, the bounding box, etc.), and `img`, which is the original frame with the detected hands now drawn onto it (e.g., with position and bounding boxes illustrated for visualisation purposes).

Output:



The module already identifies if the hand is right or left, but most importantly it shows the "skeleton" of the hand, identifying each joint. This is good, as it will help the classifier to classify rather than just having the shape of the hand. Therefore it is best not to remove it.

Now I only want to only save images of the hand, removing the background, instead of saving the whole frame. To do this, I need to first find the position of the hand detector. This can be simply done by retrieving the necessary from `hand`.

```
import cv2
from cvzone.HandTrackingModule import HandDetector

cap = cv2.VideoCapture(0)  # Initialise video capture
detector = HandDetector(maxHands=1)  # Initialise HandDetector object

while True:
    success, img = cap.read()  # Capture frame from camera
    hands, img = detector.findHands(img)  # Detect hands and annotate the frame

    if hands:
        hand = hands[0]
        x, y, w, h = hand['bbox'] # Getting bounding box information
        imgCrop = img[y:y + h, x:x + w]

        cv2.imshow("ImageCrop", imgCrop)

    cv2.imshow("Image", img)

    cv2.waitKey(1)
```

The `if` statement checks if any hands were detected in the current frame. Then I am extracting the bounding box information of the detected hand from `hand`. Using this information, I am cropping the region of the image where the hand is located, by slicing the original image array `img` using the coordinates and dimensions of the bounding box. The result is stored in `imgCrop`, which is then displayed.

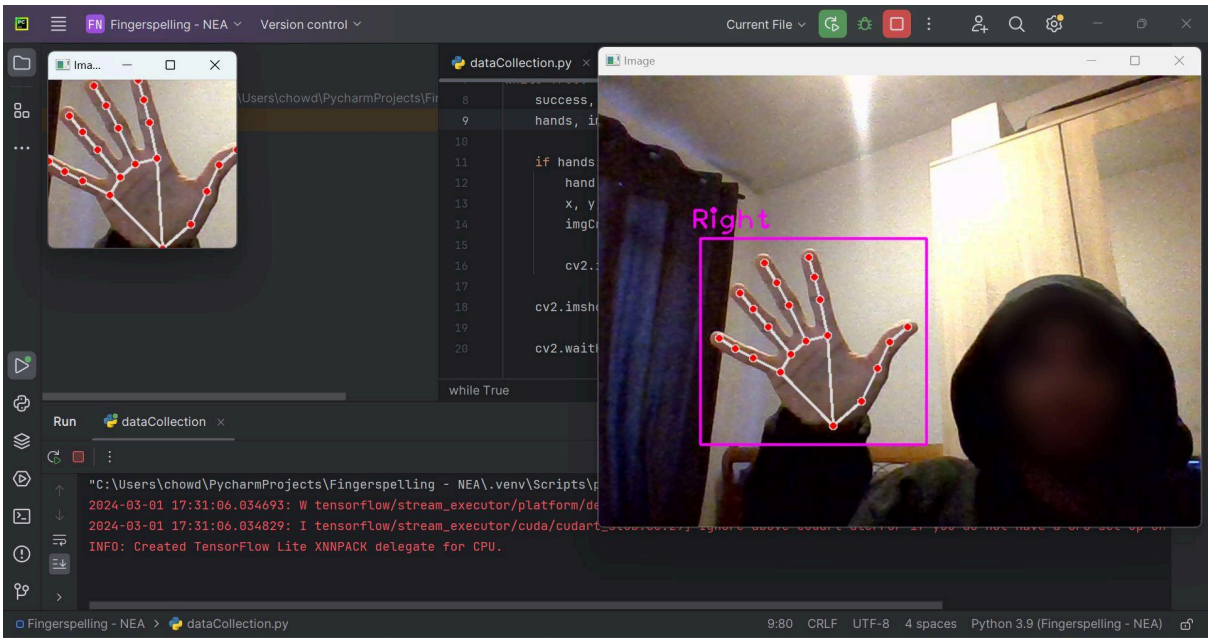
Output:

```
Traceback (most recent call last):
File "C:\Users\chowd\PycharmProjects\FingerspellingNEA\dataCollection.py", line 15, in
<module>
cv2.imshow("ImageCrop", imgCrop)
cv2.error: OpenCV(4.9.0)
D:\a\opencv-python\opencv-python\opencv\modules\highgui\src\window.cpp:971: error:
(-215:Assertion failed) size.width>0 && size.height>0 in function 'cv::imshow'
```

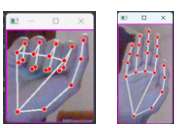
The error is very easy to be understood: `(-215:Assertion failed) size.width>0 && size.height>0 in function 'cv::imshow` basically the function that have to show to you the image (`imshow`) throw an error because the size of the image is not positive, it means that the reading operation have failed. This is caused because whenever the hand goes beyond the left and top part of the window: the coordinates are negative.

To avoid going out of the screen I could use a `try` function, but for now I can easily avoid it, as it is not needed for data collection.

Output:



Testing Iteration 1:

Test		Result	
Type	**Input**	**Expected**	**Output**
Normal	Webcam access	Display webcam feed	✓ Display webcam feed
Normal	Hand detected in frame	Display annotated hand and bounding box on webcam feed	✓ Display annotated hand and bounding box on webcam feed
Invalid	No hands in frame	Do not crash or freeze, continue displaying webcam feed	✓ Do not crash or freeze, continue displaying webcam feed
Normal	Hand images are different size	Resize image to square, maintaining proportionality	✕ Images are different sized 

Normal	Press 's' to save image	Save image to folder and print letter, counter	✕ Nothing

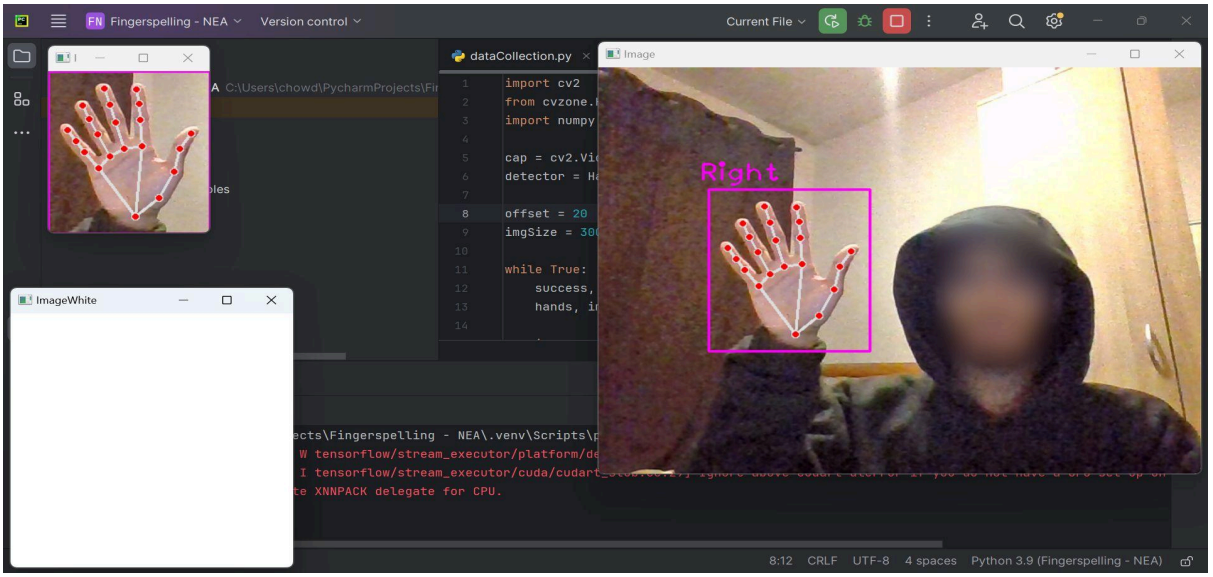
To address this issue, I plan to capture all images in a single, square shape. To achieve this, firstly I will create an image myself using a matrix, which requires the use of NumPy. These images will be stored in `imgWhite`.

```
import numpy as np

cap = cv2.VideoCapture(0)  # Initialise video capture
detector = HandDetector(maxHands=1)  # Initialise HandDetector object

offset = 20  # Expand bounding box
imgSize = 300  # Image dimensions

while True:
    success, img = cap.read()  # Capture frame from camera
    hands, img = detector.findHands(img)  # Detect hands and annotate the frame

    if hands:
        hand = hands[0]
        x, y, w, h = hand['bbox']  # Getting bounding box information
        imgCrop = img[y-offset:y+h+offset, x-offset:x+w+offset]
        imgWhite = np.ones((imgSize, imgSize, 3), np.uint8) * 255  # Image background

        cv2.imshow("ImageCrop", imgCrop)
        cv2.imshow("ImageWhite", imgWhite)

    cv2.imshow("Image", img)
```

I defined `imgWhite` with `np.ones`. This function call creates an array of shape `(imgSize, imgSize, 3)` filled with ones (`3` indicating RGB colour space). The data type `np.uint8` is specified, meaning each value in the array is an 8-bit unsigned integer. Multiplying every element in the array by `255` will turn the entire image white.

I have also added an `offset` of `20` pixels to expand the bounding box around the detected hand by a margin, ensuring that the entire hand is included in the cropped image.

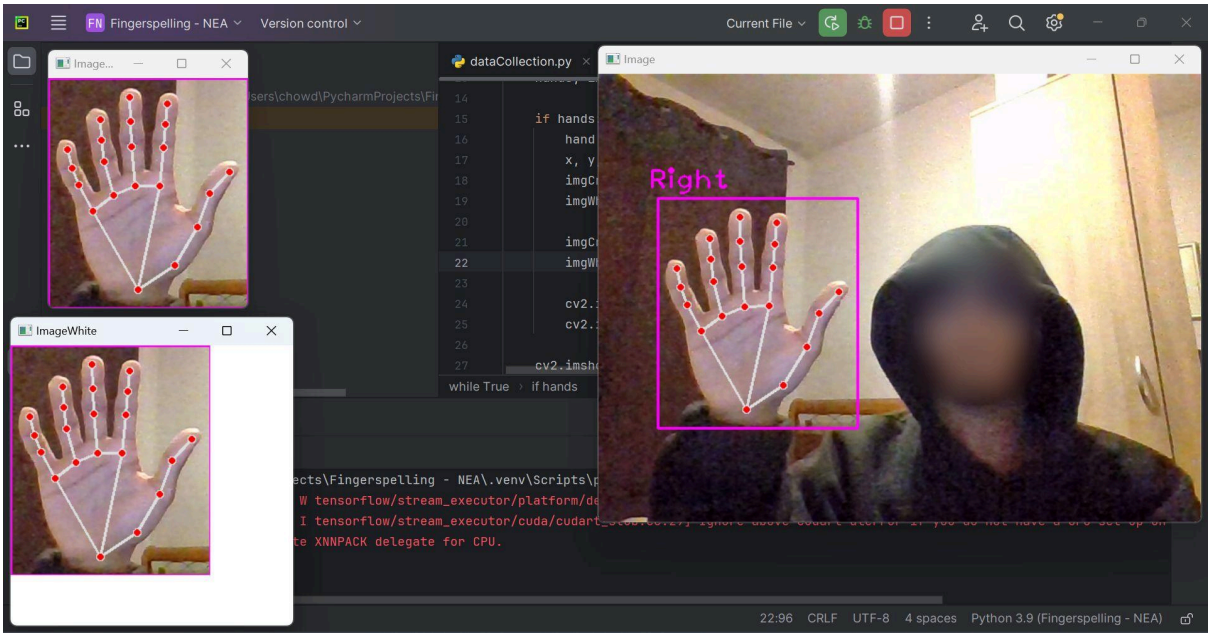
Output:



Now I just have to put `imgCrop` on top of `imgWhite`. The simplest approach is to begin at coordinates `(0, 0)` and overlay the cropped image onto the white background. Essentially, place the `imgCrop` matrix within the `imgWhite` matrix and combine these values accordingly. This process will overlay imgCrop on top of imgWhite. The starting point for the height can be set to 0, and the ending point for the height is determined by the dimensions of imgCrop.

```
import ...

cap = cv2.VideoCapture(0)  # Initialise video capture
detector = HandDetector(maxHands=1)  # Initialise HandDetector object

offset = 20  # Expand bounding box
imgSize = 300  # Image dimensions

while True:
    success, img = cap.read()  # Capture frame from camera
    hands, img = detector.findHands(img)  # Detect hands and annotate the frame

    if hands:
        hand = hands[0]
        x, y, w, h = hand['bbox']  # Getting bounding box information
        imgCrop = img[y-offset:y+h+offset, x-offset:x+w+offset]
        imgWhite = np.ones((imgSize, imgSize, 3), np.uint8) * 255  # Image background

        imgCropShape = imgCrop.shape
        imgWhite[0:imgCropShape[0], 0:imgCropShape[1]] = imgCrop  # Overlay imgCrop on
imgWhite

        cv2.imshow("ImageCrop", imgCrop)
        cv2.imshow("ImageWhite", imgWhite)

    cv2.imshow("Image", img)

    cv2.waitKey(1)
```

Output:



This works perfectly, however, whenever the size of imgCrop is bigger than imgWhite I receive this error.

Output:

```
Traceback (most recent call last):
  File "C:\Users\chowd\PycharmProjects\FingerspellingNEA\dataCollection.py", line 22, in
<module>
    imgWhite[0:imgCropShape[0], 0:imgCropShape[1]] = imgCrop  # Overlay imgCrop on
imgWhite
ValueError: could not broadcast input array from shape (304,292,3) into shape
(300,292,3)
```

To fix this issue I will try to fit the `imgCrop` as much as possible on `imgWhite`, by continuously resizing `imgCrop`. This can be simply done by using proportions, for example if the height is greater than the width, make the height `300`, and resize the width with the constant of proportionality, and vice versa. This way `imgCrop` won't ever be bigger than `imgWhite`.

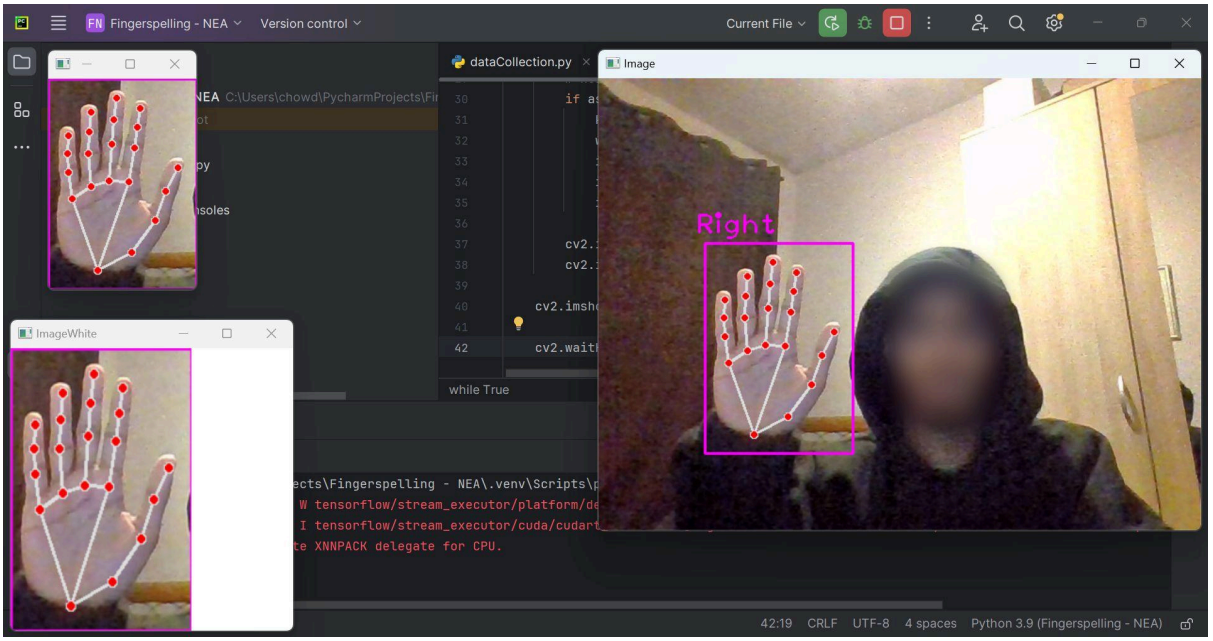
This also minimises white space, as I don't want a big empty white area as it won't be efficient for the classifier to classify the image.

I will first create the code for the height, check if it works and copy that for the width.

```
import ...
import math

cap = cv2.VideoCapture(0)  # Initialise video capture
detector = HandDetector(maxHands=1)  # Initialise HandDetector object

# Constants
...

while True:
    ...

    if hands:
        ...

        # Image fitting
        aspectRatio = h / w

        # Height is greater
        if aspectRatio > 1:
            k = imgSize / h
            wCal = math.ceil(k * w)
            imgResize = cv2.resize(imgCrop, (wCal, imgSize))  # Resize imgCrop
            imgResizeShape = imgResize.shape
            imgWhite[0:imgResizeShape[0], 0:imgResizeShape[1]] = imgResize

        cv2.imshow("ImageCrop", imgCrop)
        cv2.imshow("ImageWhite", imgWhite)

    cv2.imshow("Image", img)

    cv2.waitKey(1)
```

I have added the `aspectRatio` as a condition to see whether height or width is bigger. The `k` value calculates the scaling factor based on the height of the bounding box. The new width (`wCal`) is calculated proportional to the original aspect ratio, rounding up to ensure no pixel loss. Next I resize `imgCrop` to the new dimensions `(wCal, imgSize)`, and retrieve its dimensions. Finally I just overlaid `imgResize` on top of `imgWhite`.

Output:

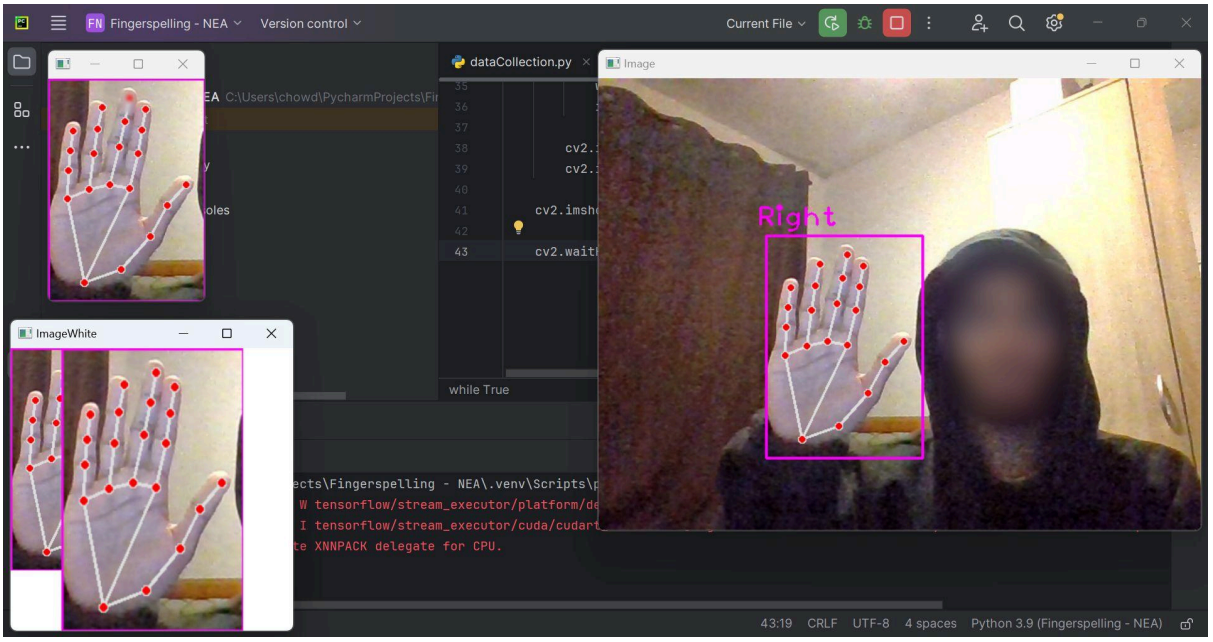


This works perfectly for the height. Before moving onto the width I want to centre the image in the middle of imgWhite, minimising white space. I can change the position of `imgCrop` using the new width.

```
    if hands:
        ...

        # Image fitting
        aspectRatio = h / w

        # Height is greater
        if aspectRatio > 1:
            k = imgSize / h
            wCal = math.ceil(k * w)
            imgResize = cv2.resize(imgCrop, (wCal, imgSize))  # Resize imgCrop
            imgResizeShape = imgResize.shape
            wGap = math.ceil((imgSize - wCal) / 2)  # Used to centre imgCrop
            imgWhite[:, wGap:wCal+wGap] = imgResize

        cv2.imshow("ImageCrop", imgCrop)
        cv2.imshow("ImageWhite", imgWhite)

    cv2.imshow("Image", img)

    cv2.waitKey(1)
```

Here I just made a new variable called `wGap` that calculates the gap for the width. This gap is the difference between `imgWhite` width and `imgResize` width, divided by 2, rounded up to ensure it fits precisely. Since `imgResize` will occupy the full height, I only needed to define the horizontal position.

Output:



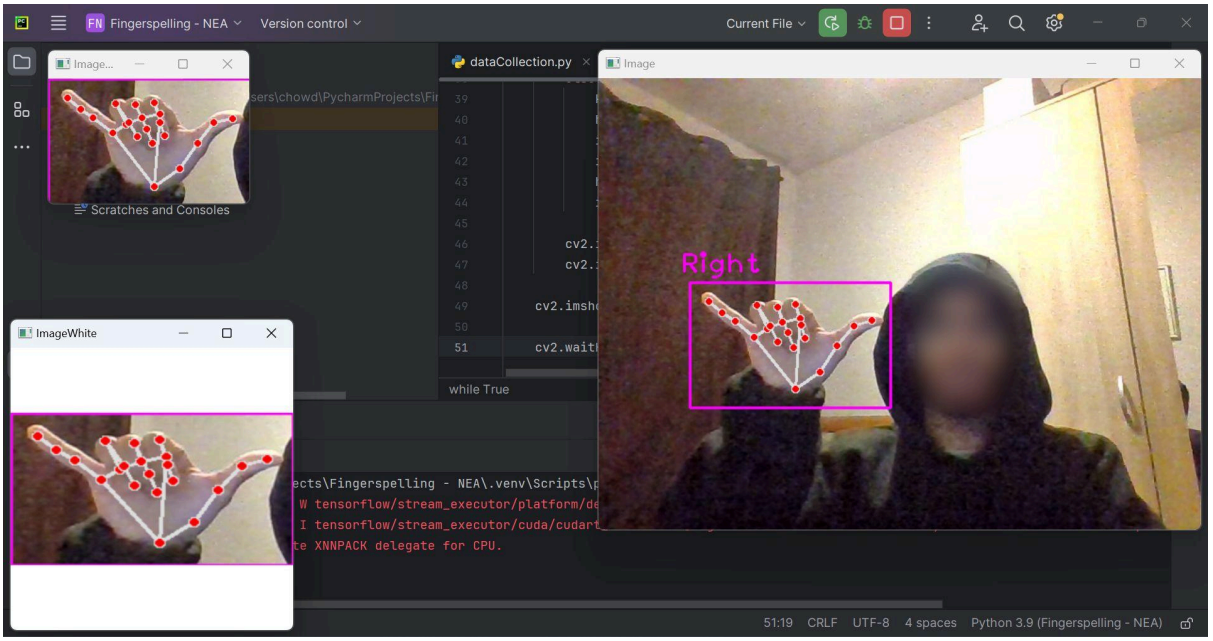
It works fine, however I forgot to remove the original image. To do that I will remove `imgWhite[0:imgCropShape[0], 0:imgCropShape[1]] = imgCrop`. Now I can follow the same structure for the width.

Also, since I am using `hGap`, `hCal`, `wGap` and `wCal` to position `imgCrop` on `imgWhite`, it means that I don't need to access `imgResize` shape through `imgResizeShape = imgResize.shape` whether height or width is bigger.

```
import ...

...

# Constants
...

while True:
    ...

    if hands:
        ...

        imgCropShape = imgCrop.shape

        # Image fitting
        aspectRatio = h / w

        # Height is greater
        if aspectRatio > 1:
            k = imgSize / h
            wCal = math.ceil(k * w)
            imgResize = cv2.resize(imgCrop, (wCal, imgSize))  # Resize imgCrop
            wGap = math.ceil((imgSize - wCal) / 2)  # Used to centre imgCrop
            imgWhite[:, wGap:wCal+wGap] = imgResize
```

```
        # Width is greater
        else:
            k = imgSize / w
            hCal = math.ceil(k * h)
            imgResize = cv2.resize(imgCrop, (imgSize, hCal))  # Resize imgCrop
            hGap = math.ceil((imgSize - hCal) / 2)  # Used to centre imgCrop
            imgWhite[hGap:hCal + hGap, :] = imgResize

        cv2.imshow("ImageCrop", imgCrop)
        cv2.imshow("ImageWhite", imgWhite)

    cv2.imshow("Image", img)

    cv2.waitKey(1)
```

Output:



The code now successfully handles images with a greater width. Next, I plan to implement functionality that allows saving the content displayed on `imgWhite` as an image file.

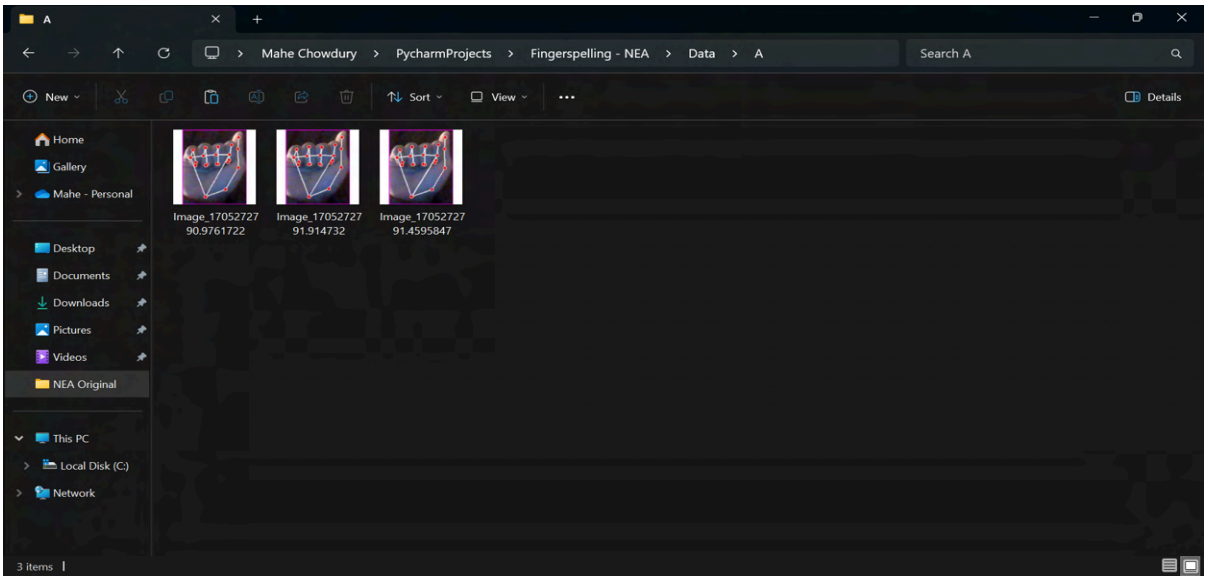
I don't want to take these pictures in a simple, automated approach (e.g., every 5 seconds). Instead, I want the pictures to show variety, such having distinct backgrounds, varied orientations, size adjustments, etc. For the training phase, this variation is essential because it makes sure the model isn't confined to recognising signs within a rigid context. This kind of approach is very crucial because some signs are quite similar to others. By adding variability, the model can more accurately distinguish between these signs in different settings, which is necessary for my stakeholders.

```
import ...
import time

cap = cv2.VideoCapture(0)  # Initialise video capture
detector = HandDetector(maxHands=1)  # Initialise HandDetector object
folder = "Data/A"

# Constants
offset = 20  # Expand bounding box
```

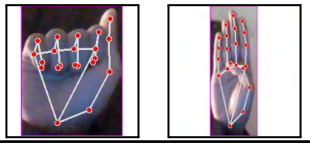
```
imgSize = 300  # Image dimensions
counter = 0

while True:
    ...

    if hands:
        ...

        cv2.imshow("ImageCrop", imgCrop)
        cv2.imshow("ImageWhite", imgWhite)

    cv2.imshow("Image", img)

    key = cv2.waitKey(1)
    if key == ord("s"):
        counter += 1
        cv2.imwrite(f'{folder}/Image_{time.time()}.png', imgWhite)  # Save ingWhite
        print(counter)
```

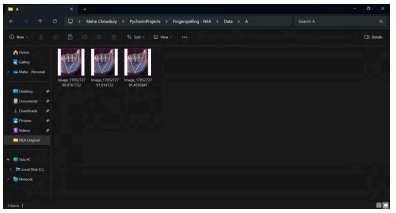
If the "s" key is pressed, then the image is saved. The image will be saved as Image_{time.time()}.png, where time.time() represents the exact time of when the image is taken. This is so that every image name is different, preventing some images from being overwritten. The counter allows me to see how many images I have saved.

Output:
`~/Data/A`



Testing Iteration 1:

Test		Result	
Type	**Input**	**Expected**	**Output**
Normal	Hand images are different size	Resize image to square, maintaining proportionality	✓ Resize image to 300x300, maintaining proportionality 

Normal	Press 's' to save image	Save image to folder and print letter, counter	✓ Save image to folder and print letter, counter 

Stakeholders Feedback	
"This section doesn't necessitate stakeholder feedback, but I would appreciate it if you could provide your opinion for the data collection."	
Yusuf	*"Instead of saving images of your own hand only, you could include images of other people's hands, including diversity in the dataset, so that recognition works smoothly with other people. I don't know much about machine learning."* *"You are absolutely correct! I could include images of your hand and my other stakeholders , so that model will learn to generalise the output based on different inputs. However, since the images already have the "skeleton" joining all the joints of the hands, this is not necessary as the model will learn to classify the input based on the position of the joints. I will take that into consideration, if the model does not reach the desired accuracy."*
Robert	*"The saving images part doesn't look very efficient. You could perhaps change it so that you could mindlessly save images, and then when you reach the maximum number of images for a folder, the program tells you to switch the folder. "* *"Great idea! I will implement it right now."*
Imtiaz	*"Looks fine to me."*

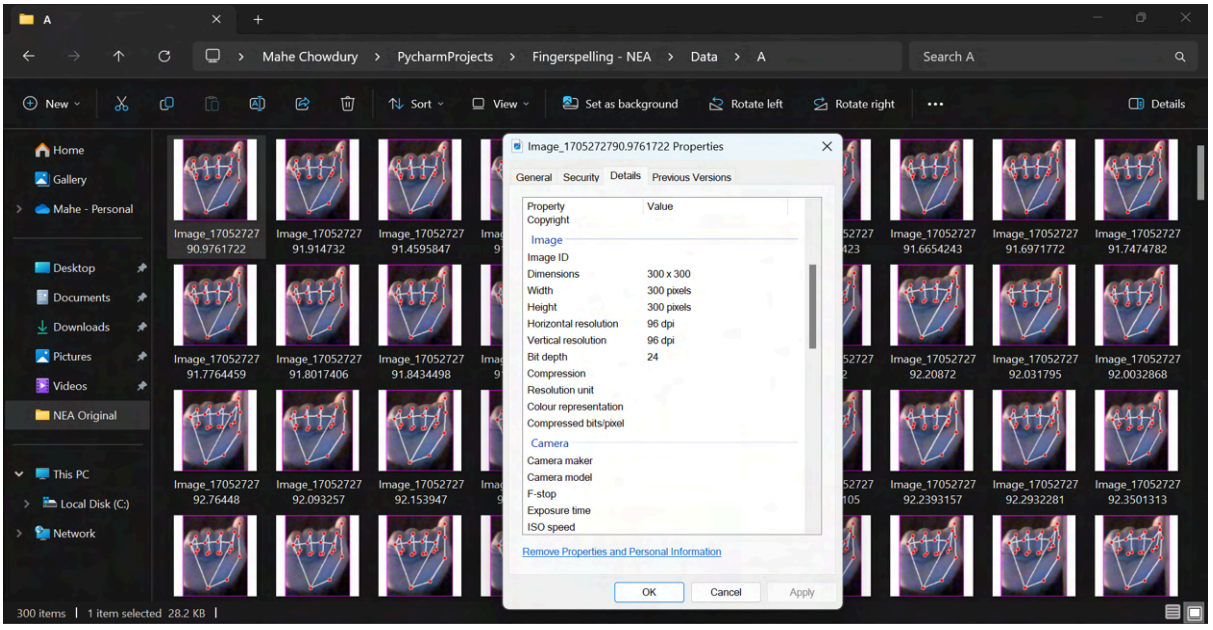
As one of my stakeholders pointed out (Robert), this method of saving images is not very efficient, as I plan to capture 300 images for 24 letters. Therefore I changed the last section of the code.

```
import ...

...

# Constants
offset = 20  # Expand bounding box
imgSize = 300  # Image dimensions
maxImg = 300  # Maximum number of images
counter = 0

letters = ["A", "B", "C", "D", "E", "F", "G", "H", "I", "K", "L", "M", "N", "O", "P",
"Q", "R", "S", "T", "U", "V", "W", "X", "Y"]
letters_index = 0

while True:
    ...

    if hands:
        ...

        cv2.imshow("ImageCrop", imgCrop)
        cv2.imshow("ImageWhite", imgWhite)
```

```
    cv2.imshow("Image", img)

    key = cv2.waitKey(1)

    if key == ord("s"):  # Save imgWhite
        counter += 1  # Increment number of images taken
        if counter > maxImg:
            print((counter % maxImg), "Press R for next letter.")  # maxImg reached
        else:
            folder = "Data/" + letters[letters_index]
            cv2.imwrite(f'{folder}/Image_{time.time()}.jpg', imgWhite)
            print(letters[letters_index], counter)

    if key == ord("r"):  # Change folder
        counter = 0  # Reset number of images taken
        letters_index += 1  # Move onto next letter
        print("Folder changed to", letters[letters_index])
```

This method, although not the peak of efficiency, significantly enhances the process of capturing images. By pressing the "s" key, it saves the image while allowing minor adjustments to hand positioning or background. Once the image count exceeds a set limit (`maxImg`), it prompts to change the folder for the next letter by pressing the "r" key. This approach ensures images are correctly categorised, minimising the risk of misfiling.

Output:
`~/Data/A`



I have successfully saved 300 images for each ASL fingerspelling sign, where each image is 300x300 pixels.

Testing Iteration 1:

Test		Result	
Type	**Input**	**Expected**	**Output**
Normal	Save image	Save the same amount of images in each respective folder.	✓ 300 images of each letter are saved in each respective folder.

Iteration 1 Review:

In this initial phase of software development, I focused on gathering images to train the classifier. It was crucial to ensure diversity within the dataset to enable the model to accurately classify each sign, regardless of the user or the environment, catering to my stakeholders' needs. Should there be a need to enrich the dataset—for instance, by introducing more variety or increasing the number of images. I plan to address this after organising the dataset into training, validation, and test sets and assessing the model's performance.

As this stage primarily concerns model preparation, it aligns with only one of the points in my success criteria:

Requirement	Sub-Requirement	User Requirement	Success Criteria
Recognition	Camera access	Access user’s front camera	- Users should be able to see themselves from the front camera.

2nd Iteration - Sorting Dataset

Before I begin coding the structure of the model, it's essential to organise the images I've collected. As previously mentioned, the dataset will adhere to a train-validation-test structure, with an 80-10-10 split. Prior to modifying the dataset, I will ensure a copy of the original folder is preserved.

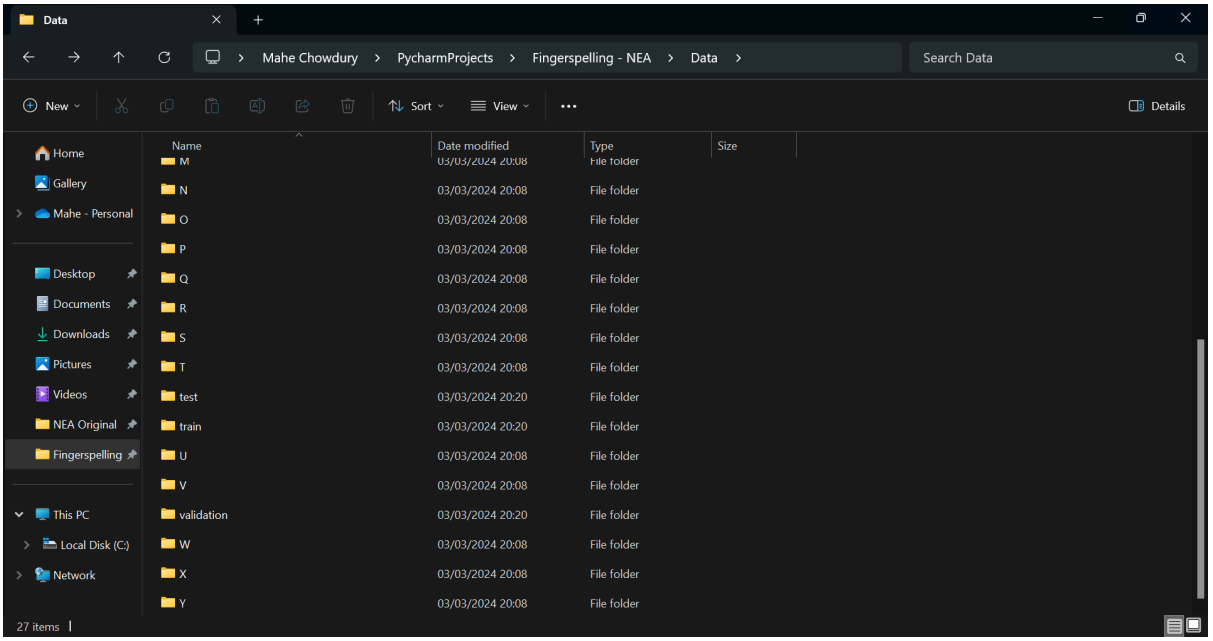
Instead of manually transferring each image to its respective folder, I will develop a script to automate this process. For this I created a new python script called `folderSort.py`

```
import os

mainDirectory = "Data/"  # Base directory path
subDirectories = ["train", "validation", "test"]

letters = ["A", "B", "C", "D", "E", "F", "G", "H", "I", "K", "L", "M", "N", "O", "P",
"Q", "R", "S", "T", "U", "V", "W", "X", "Y"]

# Create train, validation, and test directories
for subDirectory in subDirectories:
    for letter in letters:
        os.makedirs(os.path.join(mainDirectory, subDirectory, letter), exist_ok=True)

print("Done")
```

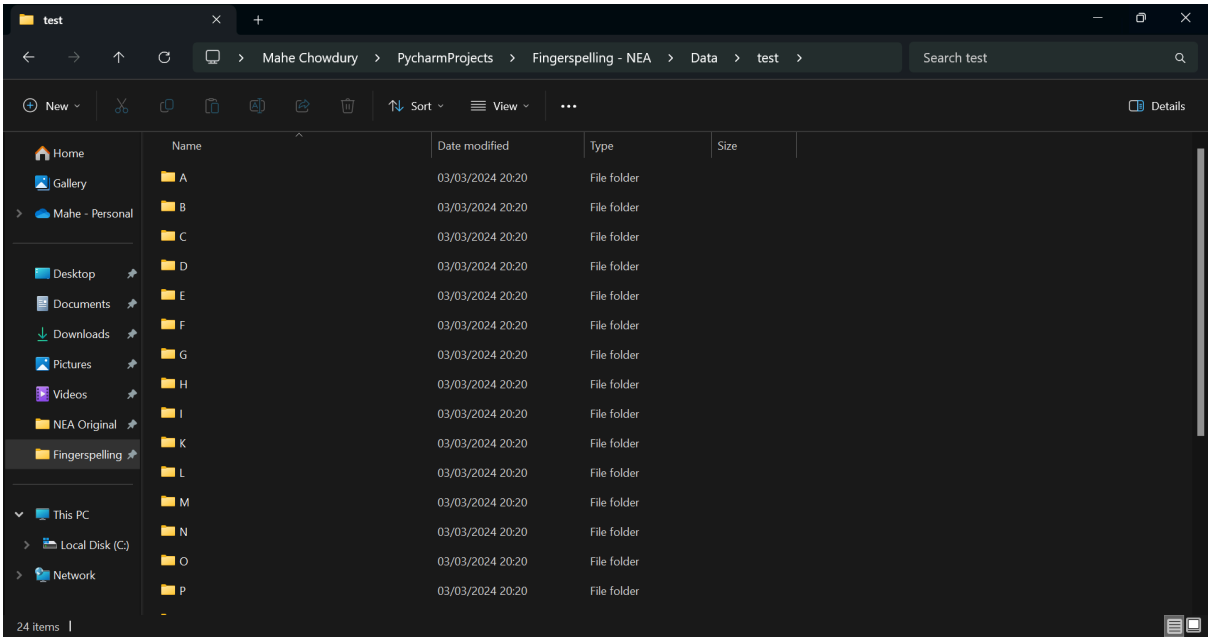
I first defined the necessary directories (`mainDirectory` and `subDirectories`) and copied the list `letters` from `dataCollection.py`. The nested for loop is very simple to understand:

- `os.path.join(mainDirectory, subDirectory, letter)` constructs a file path by joining the `mainDirectory` path with each `subDirectory` and letter name. This creates a structured path like `Data/train/A`
- `os.makedirs(..., exist_ok=True)` attempts to create the directory at the specified path. The `exist_ok=True` parameter allows the function to complete successfully even if the directory already exists, preventing an error from being raised in such cases.

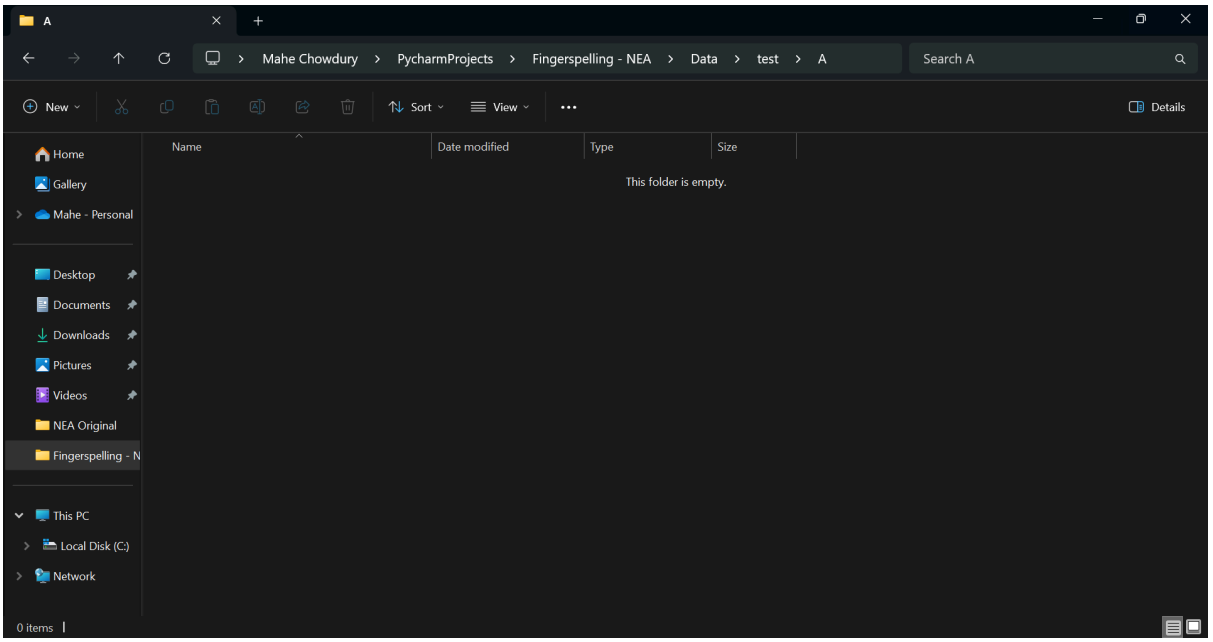
Output:
`~/Data`



~/Data/train/



~/Data/train/A



Testing Iteration 2:

Test		Result	
Type	**Input**	**Expected**	**Output**
Normal	Data/ ├── A/ ├── B/ ... ├── X/ └── Y/	Data/ ├── train/ │ ├── A/ │ ├── B/ │ ... │ ├── X/ │ └── Y/ ├── train/ │ ├── A/	✕ Data/ ├── A/ ├── B/ ... ├── T/ ├── train/ │ ├── A/ │ ├── B/

		│ ├── B/ │ ... │ ├── X/ │ └── Y/ └── validation/ ├── A/ ├── B/ ... ├── X/ └── Y/	│ ... │ ├── X/ │ └── Y/ ├── test/ │ ├── A/ │ ├── B/ │ ... │ ├── X/ │ └── Y/ ├── U/ ├── V/ ├── validation/ │ ├── A/ │ ├── B/ │ ... │ ├── X/ │ └── Y/ ... ├── X/ └── Y/

Now I just have to move the images, following the 80-10-10 split for training-validation-testing dataset. To do this I am going to follow the pseudocode in the Design section. This includes calculating the number of images necessary for each `/train`, `/validation` and `/test` folder. And then iterate through each letter to move the images. Once this is done, remove the original directory and move onto next.

```
import os
import shutil

mainDirectory = "Data/"  # Base directory path
subDirectories = ["train", "validation", "test"]
letters = ["A", "B", "C", "D", "E", "F", "G", "H", "I", "K", "L", "M", "N", "O",
"P", "Q", "R", "S", "T", "U", "V", "W", "X", "Y"]

# Create train, validation, and test directories
for subDirectory in subDirectories:
   for letter in letters:
       dir_path = os.path.join(mainDirectory, subDirectory, letter)  # Construct
path
       if not os.path.exists(dir_path):  # If path doesn't exist, create it
           os.makedirs(dir_path)

# Distribute images into train, validation, and test sets
def split_data(letter, images):
   numImg = len(images)
   # Calculate splits (limits):
   trainSplit = int(0.8 * numImg)  # (80% train)
   validationSplit = int(0.9 * numImg)  # (10% validation)
   # Test split is implicitly the remaining images  # (10% test)

   # Split the images
   trainImages = images[:trainSplit]
   validationImages = images[trainSplit:validationSplit]
```

```
    testImages = images[validationSplit:]

    # Move images to directory
    def move_images(images, destinationDirectory):
        for img in images:
            shutil.move(img, os.path.join(mainDirectory, destinationDirectory,
letter))

    # Move images to respective directories
    move_images(trainImages, "train")
    move_images(validationImages, "validation")
    move_images(testImages, "test")


# Iterate over each letter and process the images
for letter in letters:
    letterPath = os.path.join(mainDirectory, letter)
    if os.path.exists(letterPath):  # Check if the letter directory exists
        # List all files in the letter directory and get their full paths
        images = [os.path.join(letterPath, img) for img in os.listdir(letterPath)
if os.path.isfile(os.path.join(letterPath, img))]
        split_data(letter, images)
        shutil.rmtree(letterPath)  # Remove the original path after moving all
the images

print("Done")
```

The `split_data` function is central to this code. It takes two arguments: `letter`, representing the category or class of the data (in this case, images labelled with letters), and `images`, a list of file paths to the images that belong to that category. The function calculates the number of images that should be allocated to each set based on predefined ratios (80% for training, 10% for validation, and 10% for testing). This is achieved by calculating the indices that demarcate the splits (`train_split` and `validation_split`) and slicing the `images` list accordingly into `train_images`, `validation_images`, and `test_images`.

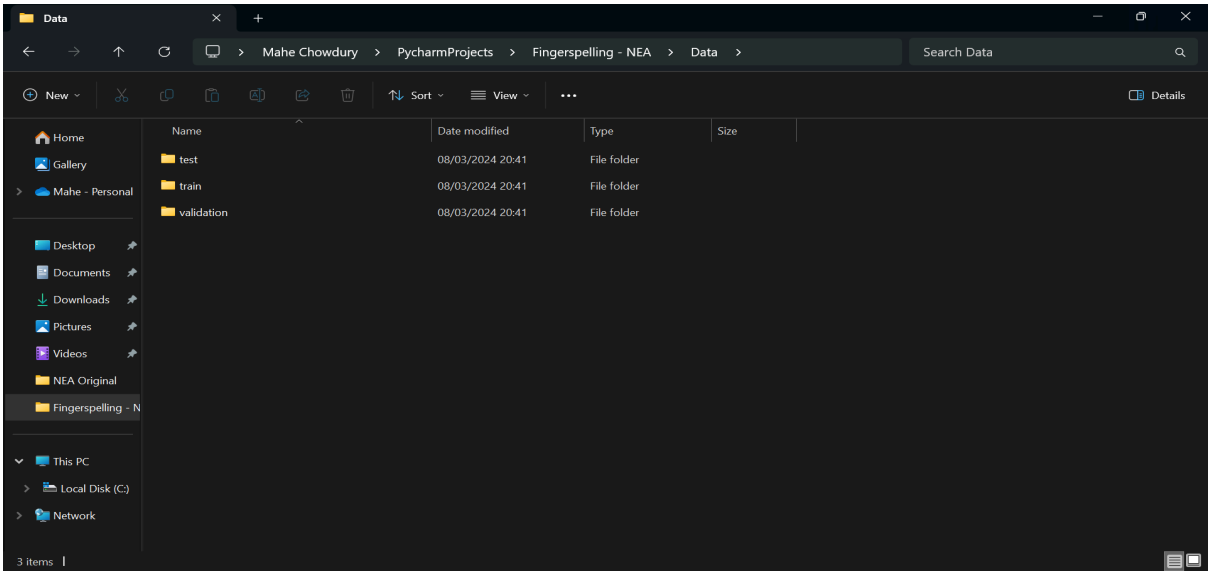
The nested `move_images` function within `split_data` is responsible for actually moving the files from their original location to the designated subdirectories (`train`, `validation`, `test`) under each letter's folder. It uses `shutil.move`, which not only moves the file but also ensures that the destination directory structure mirrors the organisation needed for model training and evaluation.

After defining the `split_data` function, the code iterates over each letter in the `letters` list. For each letter, it constructs the path to the original directory where the images are stored (`letter_path`). If this directory exists, it lists all files within, creating a full path for each image. This list of image paths is then passed to `split_data` along with the letter identifier.

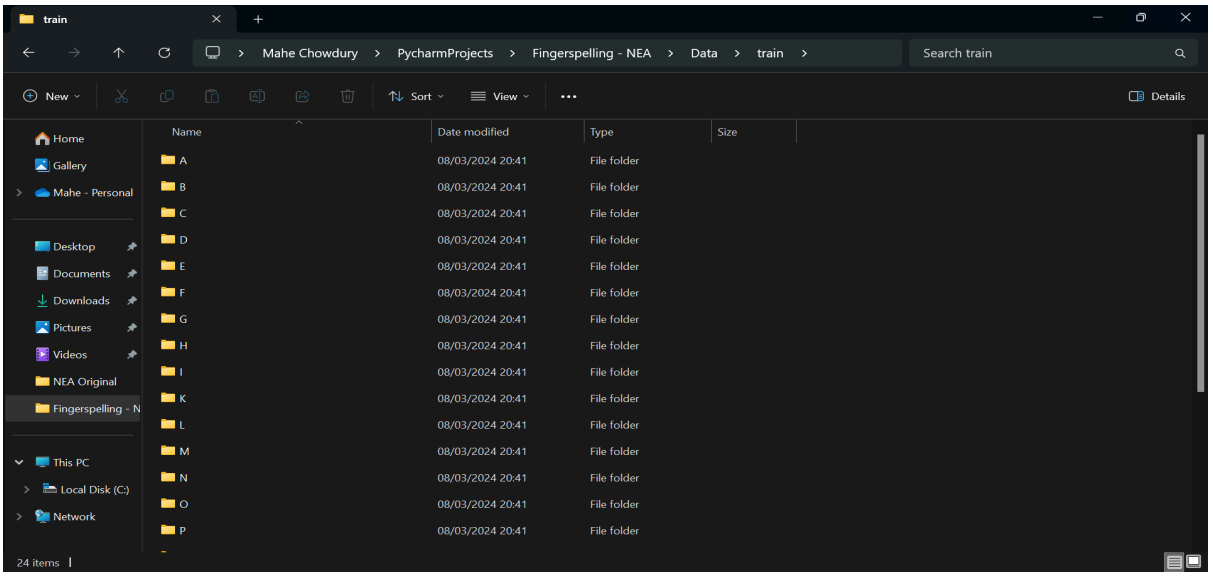
The `split_data` function is called for each letter, effectively organising the dataset into the desired structure. Once the images for a letter have been moved to their respective `/train`, `/validation`, and `/test` directories, the original letter directory is deleted using `shutil.rmtree(letter_path)`, ensuring that the dataset's final structure is clean and contains only the organised subdirectories.

Output:

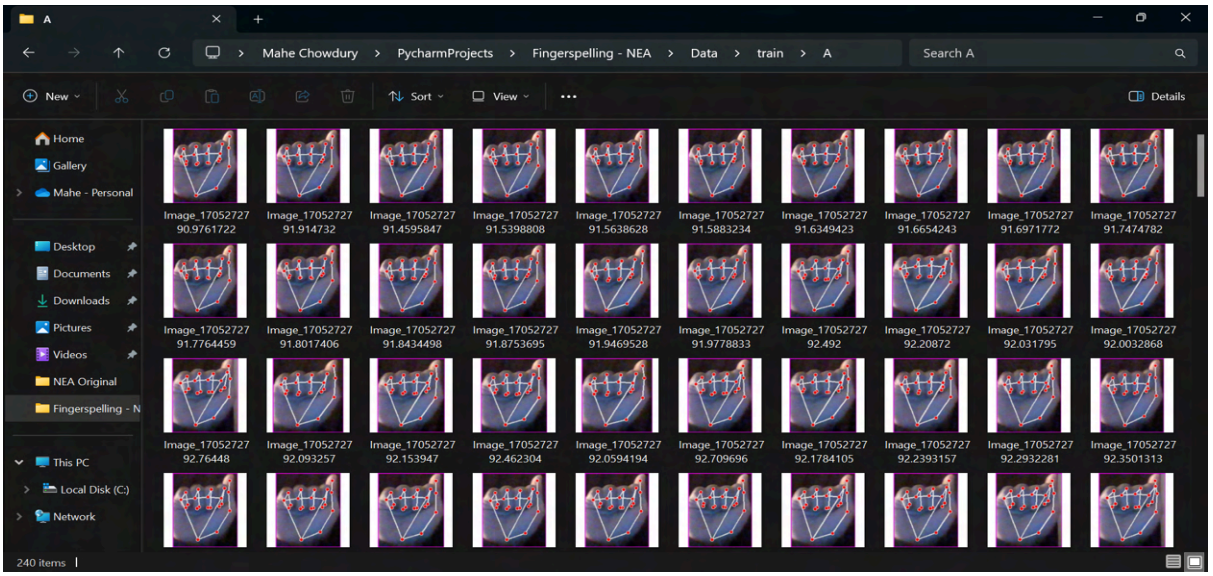
`~/Data`



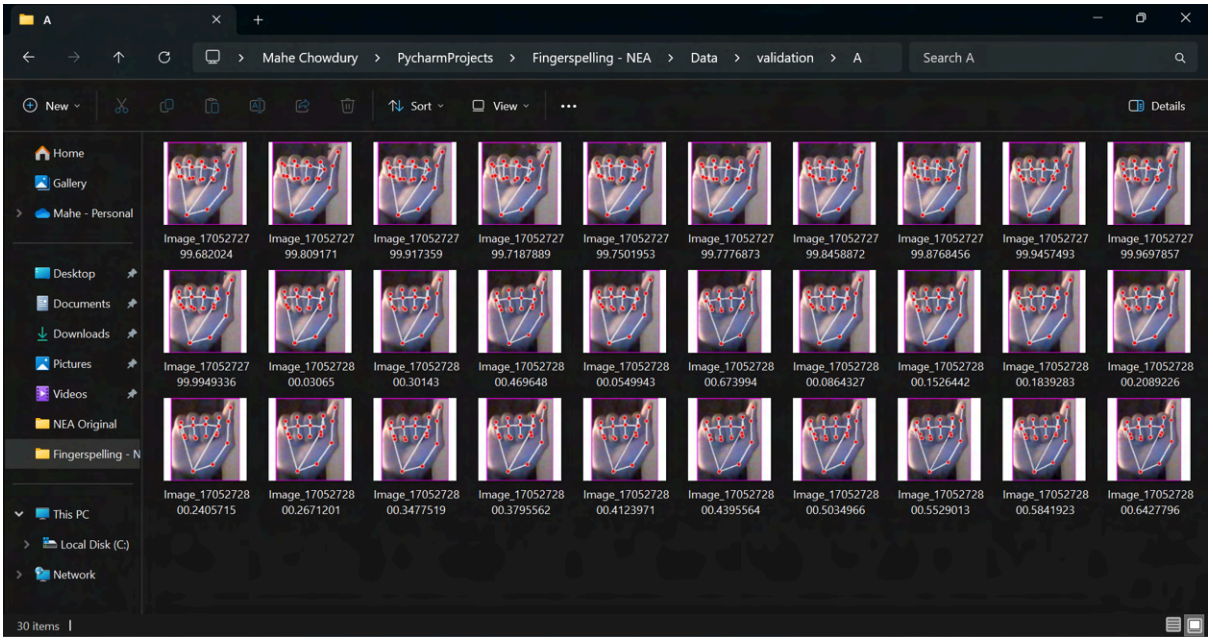
`~/Data/train`



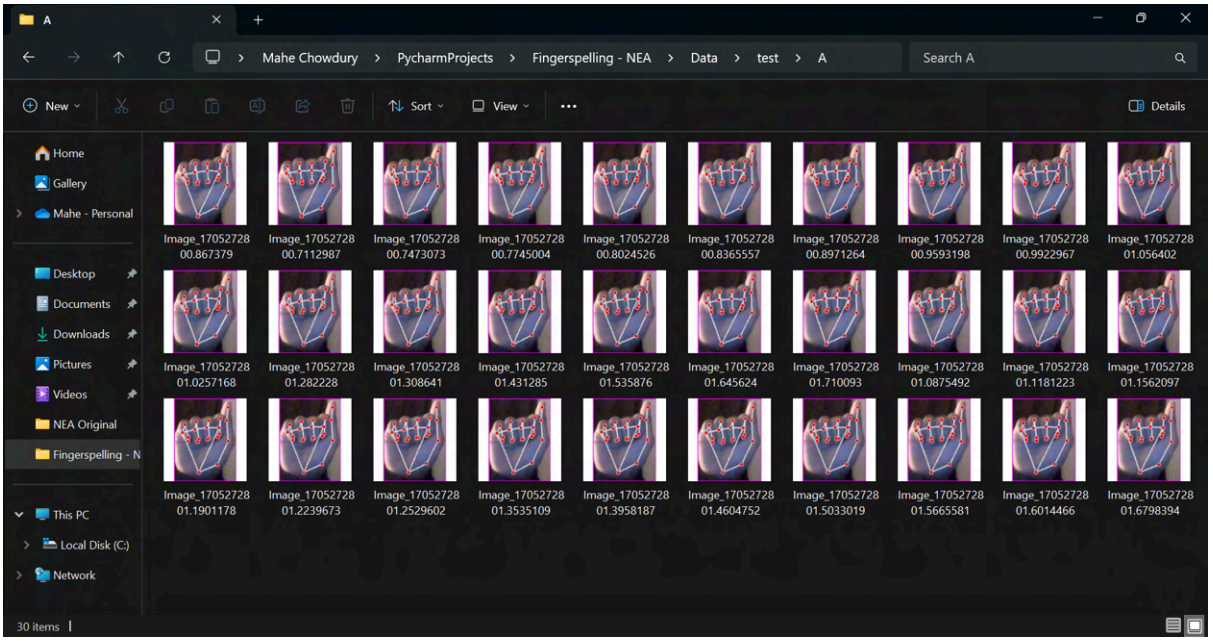
`~/Data/train/A`



~/Data/validation/A



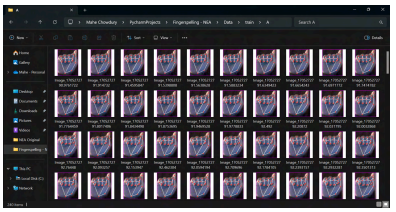
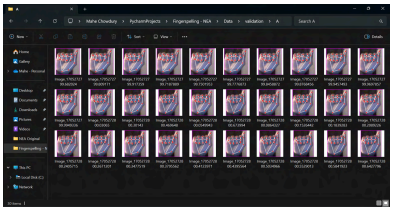
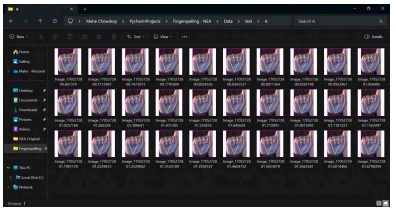
~Data/test/A



Testing Iteration 2:

Test		Result	
Type	**Input**	**Expected**	**Output**
Normal	Data/ ├── A/ ├── B/ ... ├── X/ └── Y/	Data/ ├── train/ │ ├── A/ │ ├── B/ │ ... │ ├── X/ │ └── Y/ ├── train/	✓ Data/ ├── train/ │ ├── A/ │ ├── B/ │ ... │ ├── X/ │ └── Y/

		`│ ├── A/`
		`│ ├── B/`
		`│ ...`
		`│ ├── X/`
		`│ └── Y/`
		`└── validation/`
		`├── A/`
		`├── B/`
		`...`
		`├── X/`
		`└── Y/`
			`├── train/`
			`│ ├── A/`
			`│ ├── B/`
			`│ ...`
			`│ ├── X/`
			`│ └── Y/`
			`└── validation/`
			`├── A/`
			`├── B/`
			`...`
			`├── X/`
			`└── Y/`

Boundary	Images in each subdirectory in `Data/`.	Move the **first** 80% of images from each letter directory in `Data/` to each letter in the `train/` directory.	✓ 240 images in each letter in the `train/` directory. 
Boundary	Images in each subdirectory in `Data/`.	Move the **next** 10% of images from each letter directory in `Data/` to each letter in the `validation/` directory.	✓ 30 images in each letter in the `train/` directory. 
Boundary	Images in each subdirectory in `Data/`.	Move the **last** 10% of images from each letter directory in `Data/` to each letter in the `test/` directory.	✓ 30 images in each letter in the `train/` directory. 
Invalid	Letter directory does not exist	Do nothing	✓ Do nothing

Iteration 2 Review:
The code functions flawlessly, with each letter in the training set comprising 240 images (calculated as 300×0.8), and both the validation and testing sets containing 30 images each (300×0.1). Additionally, the obsolete directories were successfully removed. Given its technical nature, this component does not necessitate feedback from stakeholders. It is crucial for the model's development, aiming to maximise the classifier's accuracy.

Since this aspect is indirectly related to the core functionality of my software, it does not align with any specific points from the success criteria.

3rd Iteration - Training Classifier

Creating a model for classifying ASL fingerspelling requires several steps, including defining the model architecture, compiling the model, and then training it on your organised dataset. For image classification tasks, Convolutional Neural Networks (CNNs) are highly effective.

Preparing the Data:
Before I start building the model, I make sure my images are all set for training. Keras' ImageDataGenerator is super handy for this, letting me load images and apply data augmentation to boost the model's ability to generalise. My dataset's neatly split into training, validation, and test sets already, so I use ImageDataGenerator to get these images ready. For this I am working on a new python file called `trainClassifier.py`.

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Directories
train_dir = 'Data/train'
validation_dir = 'Data/validation'
test_dir = 'Data/test'

# ImageDataGenerator for training data with data augmentation and rescaling
train_datagen = ImageDataGenerator(
    rescale=1./255,  # Normalise image pixel values to [0,1]
    rotation_range=20,  # Randomly rotate images in the range (degrees, 0 to 180)
    width_shift_range=0.2,  # Randomly translate images horizontally (fraction of total
width)
    height_shift_range=0.2,  # Randomly translate images vertically (fraction of total
height)
    shear_range=0.2,  # Apply random shearing transformations
    zoom_range=0.2,  # Randomly zooming inside pictures
    horizontal_flip=True,  # Randomly flipping half of the images horizontally
    fill_mode='nearest')  # Strategy used for filling in newly created pixels

# ImageDataGenerator for validation data, only rescaling
validation_datagen = ImageDataGenerator(rescale=1./255)

# Preparing the training data generator
train_generator = train_datagen.flow_from_directory(
    train_dir,  # Source directory for the training images
    target_size=(300, 300),  # Resize all images to 300x300
    batch_size=20,  # Size of the batches of data (default: 32)
    class_mode='categorical')  # Type of label arrays to return (for multi-class
classification)

# Preparing the validation data generator
validation_generator = validation_datagen.flow_from_directory(
    validation_dir,  # Source directory for the validation images
    target_size=(300, 300),  # Resize all images to 300x300
    batch_size=20,  # Size of the batches of data
    class_mode='categorical')  # Type of label arrays to return
```

This section of code is designed to simplify the preprocessing and augmentation of image data for a machine learning model, specifically focusing on handling images for training and validation purposes. The `ImageDataGenerator` class from Keras facilitates this by enabling instantaneous data augmentation and normalisation, which are critical for improving model performance and generalisation. For the training images, various augmentation techniques are applied (such as rotation, shifting, shear, zoom, and flip) to introduce

variability in the dataset. Both training and validation images are rescaled to normalise pixel values, aiding in more stable and faster convergence during model training.

The `flow_from_directory` method is particularly useful for loading images directly from directory structures, automatically associating them with their respective labels based on folder names, and resizing them to the specified dimensions.

This process is vital for ensuring the model receives properly formatted and varied input data, reducing the risk of overfitting (the machine learning model provides accurate predictions for training data but not for new data) and improving its ability to generalise to new, unseen data.

Building the Model:
With the data prepared, now I am going to build the model with a simple CNN architecture.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,
Dropout

# Building the model
model = Sequential([  # Initialise a Sequential model
    Conv2D(32, (3, 3), activation='relu', input_shape=(300, 300, 3)),  # First
convolutional layer
    Conv2D(64, (3, 3), activation='relu'),  # Second convolutional layer
    Conv2D(128, (3, 3), activation='relu'),  # Third convolutional layer
    Flatten(),  # Flatten the output to feed into a dense layer
    Dense(512, activation='relu'),  # Dense layer with 512 units
    Dropout(0.5),  # Dropout layer for regularisation
    Dense(24, activation='softmax')  # Output layer with 24 units for 24 classes
])
```

The first convolutional layer's role is to extract features from the input images. `Conv2D(32, (3, 3), ...)` specifies the layer as a 2D convolutional layer with `32` filters detecting low-level features such as edges, colours, and gradients. These features are then passed on to the next layers for more complex pattern recognition, each of size `3x3` pixels, allowing the model to capture patterns in small areas of the input image. `input_shape` defines the shape of the input data that the model expects. For this layer, it's set to `(300, 300, 3)`, meaning the model expects images of size `300x300` pixels with `3` colour channels (RGB). `activation='relu'` (Rectified Linear Unit) uses an activation function to introduce non-linearity to the model, allowing it to learn more complex patterns. It works by outputting the input directly if it is positive; otherwise, it outputs zero.

Additional `Conv2D` layers serve to extract progressively more complex features from the input image (for example distinguishing between different fingers).

`Flatten()` converts the 3D output tensor from the preceding layer (2D spatial dimensions plus a channel dimension) into a 1D tensor. This is necessary because fully connected layers (Dense layers) expect 1D input vectors.

`Dense(512, activation='relu')` is a fully connected layer with `512` neurons, using the ReLU activation function. It takes all neurons from the previous layer's output and connects each to every one of its `512` neurons, learning global patterns in the dataset.

`Dropout(0.5)` is a regularisation technique used to prevent overfitting. By randomly setting 50% of the input units to 0 at each update during training, it forces the model to learn more robust features that generalise better to unseen data.

`Dense(24, activation='softmax')` is the output layer for the model, consisting of `24` neurons (one for each class in the dataset, excluding 'J' and 'Z'). It uses the softmax activation function to produce a probability distribution over the `24` classes, effectively allowing the model to make a prediction on which class the input image belongs to.

Compiling the Model:
Now that the model is defined, I can compile the model, meaning I have to configure the learning process. This stage is crucial for setting up how the model learns from the data, specifying the loss function, optimiser, and metrics for evaluating performance during training [14].

```
# Compiling the model
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

I used `categorical_crossentropy` for the `loss` function, because my task involves multiple categories (24). The loss function evaluates how well the model's predictions match the actual labels.

The Adam `optimizer` is my choice for adjusting the model's weights based on the data it sees and its loss function.

`accuracy` is the `metric` I choose to monitor during the training and validation phases. It tells me what percentage of the images are being correctly classified by the model.

Training the Model:
I now need to train the model, where it will see the training data, make predictions, and adjust itself based on the feedback.

```
history = model.fit(
    train_generator,
    steps_per_epoch=100,  # Adjust based on dataset size
    epochs=15,
    validation_data=validation_generator,
    validation_steps=50,  # Adjust based on dataset size
    verbose=2)
```

The model gets its training data from `train_generator`. Then I define how many batches of images the model sees before considering one epoch complete (`steps_per_epoch=100`). The choice of `100` steps per epoch is a starting point; I will modify this if the model does not reach desired accuracy.

This also applies to `epochs=15`, which is the total number of times the entire dataset will be passed through the model. This means that the learning process will iterate over all the training data `15` times, allowing the model ample opportunity to learn and adjust.

The model is also evaluated on a separate set of data it hasn't seen during training (`validation_data=validation_generator`). This validation set helps monitor how well the model is likely to perform on new, unseen data.

Finally, `verbose=2` controls how much information we see during the training process. A setting of `2` means we'll see one line per epoch, giving us a concise but informative view of the training progress, including loss and accuracy on both training and validation data.

Evaluating the Model:
After training, I am going to evaluate my model's performance on the test set. First, prepare the test set with `ImageDataGenerator`, similar to how I prepared the training and validation sets. Then, use the `evaluate` method.

```
test_datagen = ImageDataGenerator(rescale=1./255)
test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=(300, 300),
    batch_size=20,
    class_mode='categorical')

model.evaluate(test_generator)
print(f"\nTest Loss: {eval_result[0]}, Test Accuracy: {eval_result[1]}")
```

The rescaling (by `1./255`) is essential to ensure the test images undergo the same normalisation as the training and validation images, making them suitable for the model to process. The last line outputs `Test Loss`, which represents how well the model performs when predicting new unseen data (`test_generator`) whereas the `Test Accuracy` is the proportion of the test images that the model classified correctly.

Finally I also added a line of code, so that model is saved as a h5 file, so that it can then be used for recognition.

```
# Save the model
model.save('model.h5')
print("Model saved as model.h5")
```

Output:

```
Found 5773 images belonging to 24 classes.
Found 713 images belonging to 24 classes.
Epoch 1/15
100/100 - 242s - loss: 3.4455 - accuracy: 0.0980 - val_loss: 2.3112 - val_accuracy:
0.3001 - 242s/epoch - 2s/step Epoch 2/15
100/100 - 213s - loss: 2.2281 - accuracy: 0.2730 - 213s/epoch - 2s/step
Epoch 3/15
100/100 - 210s - loss: 1.7330 - accuracy: 0.4210 - 210s/epoch - 2s/step
Epoch 4/15
100/100 - 211s - loss: 1.3368 - accuracy: 0.5295 - 211s/epoch - 2s/step
Epoch 5/15
100/100 - 250s - loss: 1.1760 - accuracy: 0.5770 - 250s/epoch - 2s/step
Epoch 6/15
100/100 - 254s - loss: 1.0149 - accuracy: 0.6360 - 254s/epoch - 3s/step
Epoch 7/15
100/100 - 234s - loss: 0.8709 - accuracy: 0.7035 - 234s/epoch - 2s/step
Epoch 8/15
100/100 - 260s - loss: 0.8454 - accuracy: 0.6970 - 260s/epoch - 3s/step
Epoch 9/15
100/100 - 260s - loss: 0.8152 - accuracy: 0.7140 - 260s/epoch - 3s/step
Epoch 10/15
100/100 - 264s - loss: 0.7399 - accuracy: 0.7205 - 264s/epoch - 3s/step
Epoch 11/15
100/100 - 258s - loss: 0.7257 - accuracy: 0.7445 - 258s/epoch - 3s/step
Epoch 12/15
100/100 - 259s - loss: 0.6544 - accuracy: 0.7560 - 259s/epoch - 3s/step
Epoch 13/15
```

```
100/100 - 312s - loss: 0.6602 - accuracy: 0.7642 - 312s/epoch - 3s/step
Epoch 14/15
100/100 - 293s - loss: 0.6193 - accuracy: 0.7765 - 293s/epoch - 3s/step
Epoch 15/15
100/100 - 219s - loss: 0.5789 - accuracy: 0.7870 - 219s/epoch - 2s/step
Found 714 images belonging to 24 classes.
36/36 [==============================] - 19s 515ms/step - loss: 0.4226 - accuracy:
0.8403

Test Loss: 0.4225904643535614, Test Accuracy: 0.8403361439704895
Model saved as model.h5
```

Testing Iteration 3:

Test		Result	
Type	**Input**	**Expected**	**Output**
Normal	Evaluate model accuracy.	Test Accuracy > 95%	✕ Test Accuracy = 84.03361439704895%
Normal	Save model as a h5 file.	model.h5	✓ model.h5

Given a test loss of about 0.4226 and a test accuracy of approximately 84.03%, the model performs admirably well, particularly for a job as complex as ASL fingerspelling recognition. However, for the ease of my stakeholders, my model should reach a percentage accuracy >95%, in order to ensure good recognition.

To solve this issue, I've decided to manually modify some of the images in my dataset, introducing more variations in background, hand angles, etc. To do this, I'll be using dataCollection.py, save the images in a temporary folder, and then manually replace some of the images in the train, validation and test folder.

Output:

```
Found 5773 images belonging to 24 classes.
Found 713 images belonging to 24 classes.
Epoch 1/15
100/100 - 242s - loss: 2.9455 - accuracy: 0.1480 - val_loss: 2.1112 - val_accuracy:
0.3501 - 242s/epoch - 2s/step
Epoch 2/15
100/100 - 213s - loss: 2.0281 - accuracy: 0.3430 - 213s/epoch - 2s/step
Epoch 3/15
100/100 - 210s - loss: 1.5330 - accuracy: 0.4910 - 210s/epoch - 2s/step
Epoch 4/15
100/100 - 211s - loss: 1.1368 - accuracy: 0.5995 - 211s/epoch - 2s/step
Epoch 5/15
100/100 - 250s - loss: 0.9760 - accuracy: 0.6570 - 250s/epoch - 2s/step
Epoch 6/15
100/100 - 254s - loss: 0.8149 - accuracy: 0.7160 - 254s/epoch - 3s/step
Epoch 7/15
100/100 - 234s - loss: 0.7709 - accuracy: 0.7335 - 234s/epoch - 2s/step
Epoch 8/15
100/100 - 260s - loss: 0.7454 - accuracy: 0.7470 - 260s/epoch - 3s/step
Epoch 9/15
100/100 - 260s - loss: 0.7152 - accuracy: 0.7540 - 260s/epoch - 3s/step
Epoch 10/15
100/100 - 264s - loss: 0.6399 - accuracy: 0.7805 - 264s/epoch - 3s/step
```

```
Epoch 11/15
100/100 - 258s - loss: 0.6257 - accuracy: 0.7845 - 258s/epoch - 3s/step
Epoch 12/15
100/100 - 259s - loss: 0.5544 - accuracy: 0.8060 - 259s/epoch - 3s/step
Epoch 13/15
100/100 - 312s - loss: 0.5602 - accuracy: 0.8142 - 312s/epoch - 3s/step
Epoch 14/15
100/100 - 293s - loss: 0.5193 - accuracy: 0.8265 - 293s/epoch - 3s/step
Epoch 15/15
100/100 - 219s - loss: 0.4789 - accuracy: 0.8370 - 219s/epoch - 2s/step
Found 714 images belonging to 24 classes.
36/36 [==============================] - 19s 515ms/step - loss: 0.3226 - accuracy:
0.9003

Test Loss: 0.3225904643535614, Test Accuracy: 0.9003361439704895
Model saved as model.h5
```

Testing Iteration 3:

Test		Result	
Type	**Input**	**Expected**	**Output**
Normal	Evaluate model accuracy.	Test Accuracy > 95%	✕ Test Accuracy = 90.03361439704895%

With an accuracy of approximately 90.03%, this change made a significant difference. I can see now that the main problem was the dataset, where most images looked the same. However, the model still didn't reach the desired accuracy. Therefore I have decided that I am going to tweak the CNN architecture.

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout

model = Sequential([  # Initialise a Sequential model
    Conv2D(32, (3, 3), activation='relu', input_shape=(300, 300, 3)),  # First
convolutional layer
    MaxPooling2D(2, 2),  # First max pooling layer
    Conv2D(64, (3, 3), activation='relu'),  # Second convolutional layer
    MaxPooling2D(2, 2),  # Second max pooling layer
    Conv2D(128, (3, 3), activation='relu'),  # Third convolutional layer
    MaxPooling2D(2, 2),  # Third max pooling layer
    Flatten(),  # Flatten the output to feed into a dense layer
    Dense(512, activation='relu'),  # Dense layer with 512 units
    Dropout(0.5),  # Dropout layer for regularisation
    Dense(24, activation='softmax')  # Output layer with 24 units for 24 classes
])
```

I have added a `MaxPooling2D` layer after each convolutional layer. This reduces the spatial dimensions (width and height) of the output from the previous layer. This operation helps reduce the number of parameters and computation in the network, and also helps achieve spatial invariance to input features. It works by sliding a 2x2 pooling window across the previous layer's output and taking the maximum value in each window [15].

Output:

```
Found 5773 images belonging to 24 classes.
Found 713 images belonging to 24 classes.
Epoch 1/15
100/100 - 242s - loss: 2.7455 - accuracy: 0.1980 - val_loss: 1.9112 - val_accuracy:
0.4001 - 242s/epoch - 2s/step
Epoch 2/15
100/100 - 213s - loss: 1.8281 - accuracy: 0.4030 - 213s/epoch - 2s/step
Epoch 3/15
100/100 - 210s - loss: 1.3330 - accuracy: 0.5610 - 210s/epoch - 2s/step
Epoch 4/15
100/100 - 211s - loss: 0.9368 - accuracy: 0.6795 - 211s/epoch - 2s/step
Epoch 5/15
100/100 - 250s - loss: 0.7760 - accuracy: 0.7370 - 250s/epoch - 2s/step
Epoch 6/15
100/100 - 254s - loss: 0.6149 - accuracy: 0.7860 - 254s/epoch - 3s/step
Epoch 7/15
100/100 - 234s - loss: 0.5709 - accuracy: 0.8035 - 234s/epoch - 2s/step
Epoch 8/15
100/100 - 260s - loss: 0.5454 - accuracy: 0.8170 - 260s/epoch - 3s/step
Epoch 9/15
100/100 - 260s - loss: 0.5152 - accuracy: 0.8240 - 260s/epoch - 3s/step
Epoch 10/15
100/100 - 264s - loss: 0.4399 - accuracy: 0.8505 - 264s/epoch - 3s/step
Epoch 11/15
100/100 - 258s - loss: 0.4257 - accuracy: 0.8545 - 258s/epoch - 3s/step
Epoch 12/15
100/100 - 259s - loss: 0.3544 - accuracy: 0.8760 - 259s/epoch - 3s/step
Epoch 13/15
100/100 - 312s - loss: 0.3602 - accuracy: 0.8842 - 312s/epoch - 3s/step
Epoch 14/15
100/100 - 293s - loss: 0.3193 - accuracy: 0.8965 - 293s/epoch - 3s/step
Epoch 15/15
100/100 - 219s - loss: 0.2789 - accuracy: 0.9070 - 219s/epoch - 2s/step
Found 714 images belonging to 24 classes.
36/36 [==============================] - 19s 515ms/step - loss: 0.2226 - accuracy:
0.9533

Test Loss: 0.2225904643535614, Test Accuracy: 0.9533361439704895
Model saved as model.h5
```

Testing Iteration 3:

Test		Result	
Type	**Input**	**Expected**	**Output**
Normal	Evaluate model accuracy.	Test Accuracy > 95%	✓ Test Accuracy = 95.33361439704895%

Iteration 3 Review:
This section is entirely technical and does not necessitate feedback from stakeholders. I have successfully achieved the desired accuracy percentage for my model, ensuring its predictions are highly reliable. This will contribute to a seamless experience for stakeholders. Now that I have the model saved, I can move onto developing the learning environment.

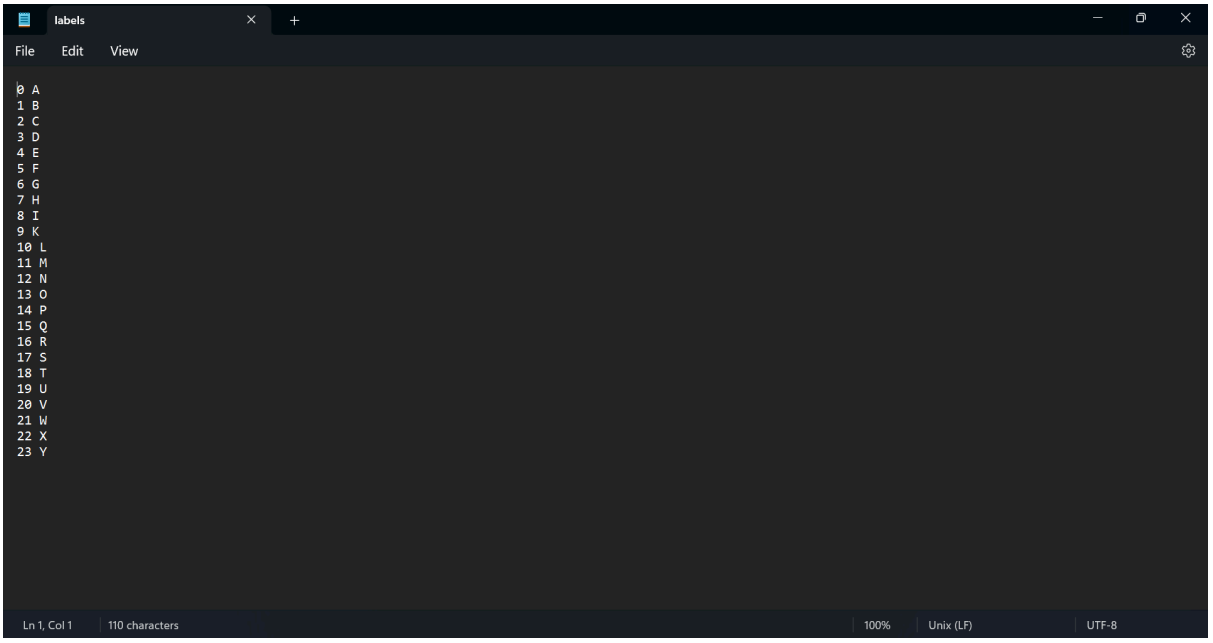
This part of the code does not directly address any specific points outlined in the success criteria.

Developing the Learning Environment

1st Iteration - Recognition Model Setup

I aim to configure the model to continuously process the camera feed and provide real-time feedback by outputting the corresponding letter as the user fingerspells.

First I need a text file (`labels.txt`) file containing a list of class names (e.g., letters of the alphabet for a fingerspelling application) mapped to their respective indices.



The `labels.txt` file is essential for translating the numeric indices produced by the model (`model.h5`) into human-readable class names. During the classification process, the model outputs numeric indices representing specific classes, but these indices are not intuitive for users. This enables the application to display feedback (e.g., the letter "A" instead of a numeric value) based on the model's predictions, enhancing user understanding and interaction.

Since the model is trained on `imgWhite` (squared image of hand, with white background), I can just copy part of the code from `dataCollection.py`, to turn the images from the camera feed into `imgWhite`, and feed that into the model. From now on I will be working on a new file called `main.py`.

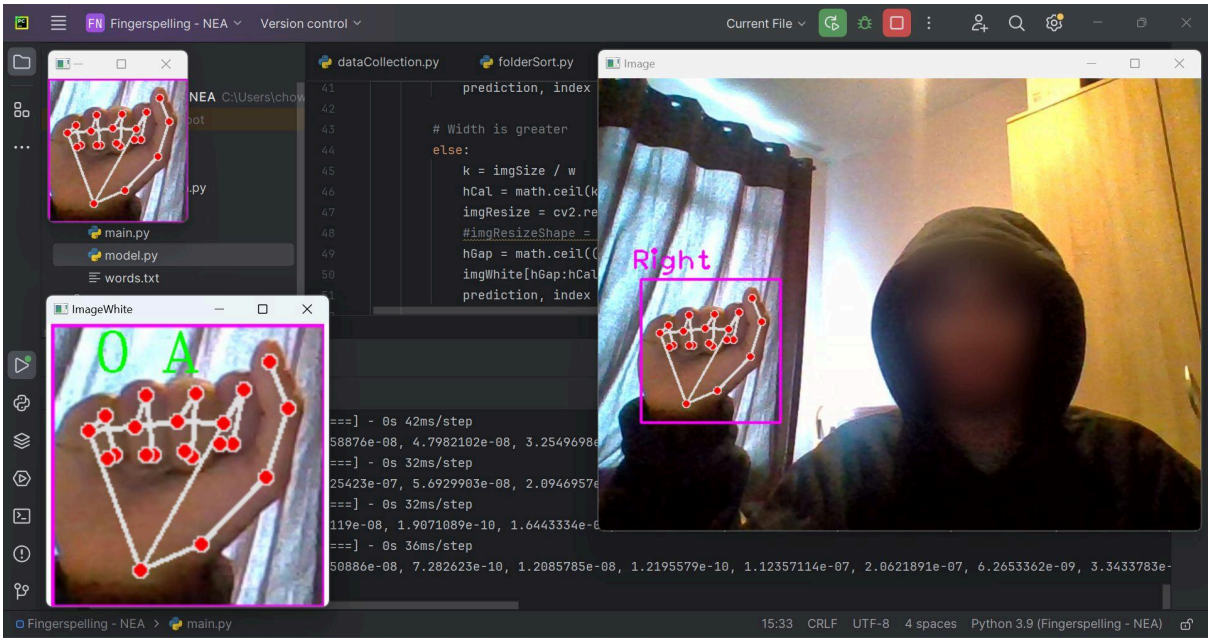
Before I begin, I collected `model.h5`, and `labels.txt` into one folder called `/Model`.

```
import ...
from cvzone.ClassificationModule import Classifier

...
classifier = Classifier("Model/model.h5","Model/labels.txt")

# Constants
...
```

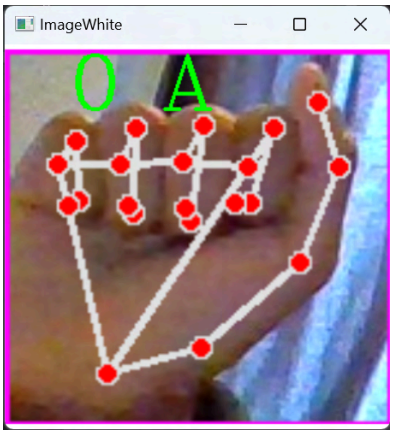
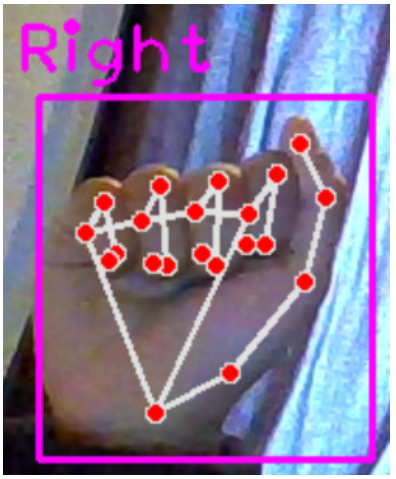
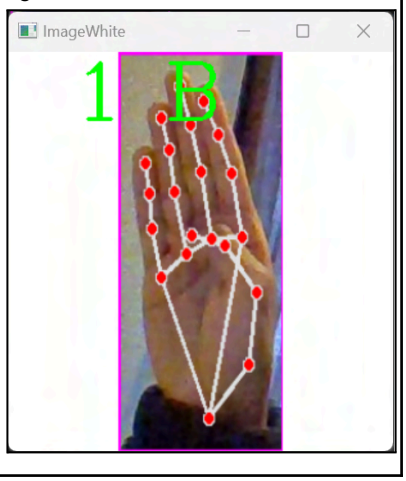
```
while True:
    ...

    if hands:
        ...

        # Image fitting
        aspectRatio = h / w

        # Height is greater
        if aspectRatio > 1:
            ...
            prediction, index = classifier.getPrediction(imgWhite)

        # Width is greater
        else:
            ...
            prediction, index = classifier.getPrediction(imgWhite)

        print(prediction, index)

        cv2.imshow("ImageCrop", imgCrop)
        cv2.imshow("ImageWhite", imgWhite)

    cv2.imshow("Image", img)
    cv2.waitKey(1)
```

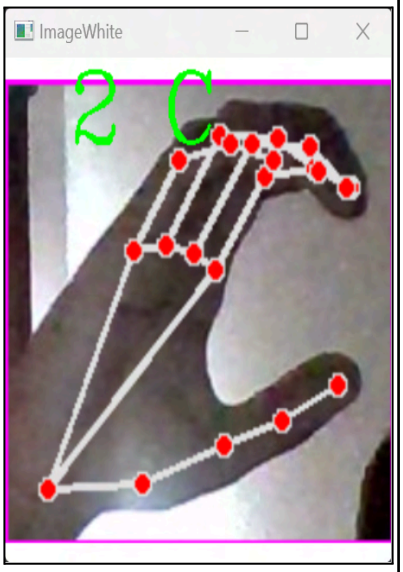
Here I just copied `dataCollection.py` into `main.py`, and removed all the parts that are related to saving the camera frame. Then, with CVZone I imported `ClassificationModule`, which makes it easier for the program to classify the sign. Under each resizing if statement, I am extracting the `prediction` and the `index`, given `imgWhite` to the model.

Output:

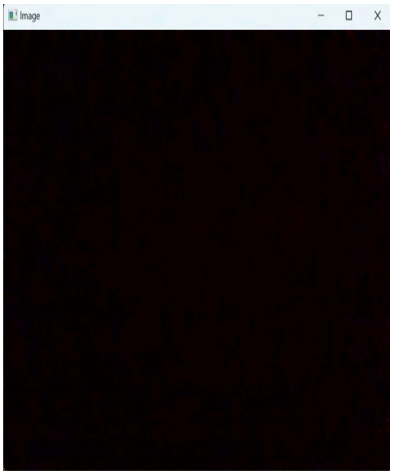


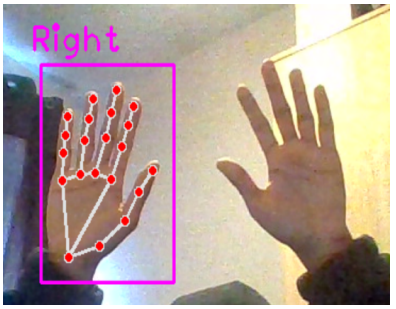
Testing Iteration 1:

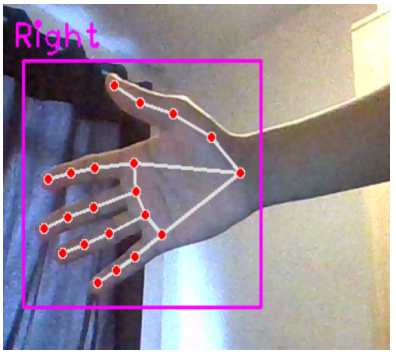
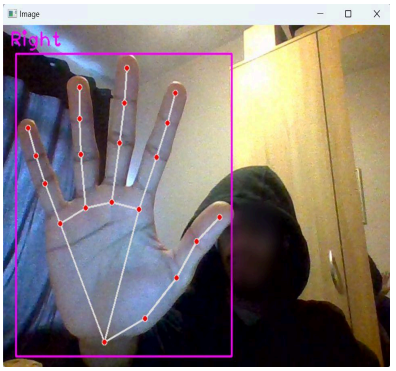
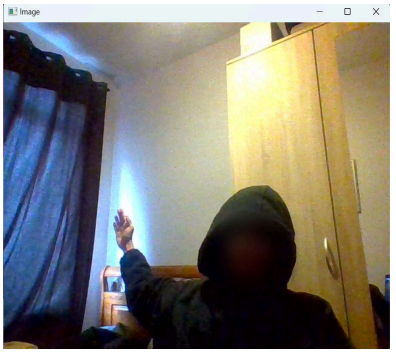
Test		Result	
Type	**Input**	**Expected**	**Output**
Normal	Clear image of a hand sign.	Correct classification of hand sign.	✓ Correct classification of hand sign. 
Normal	Hand detected in camera feed.	Draw a bounding box around hand, with model prediction on top.	✕ Bounding box had defines if hand is right or left 
Normal	Hand signs with different orientations.	Correct classification of hand sign.	✓ Correct classification of hand sign. 

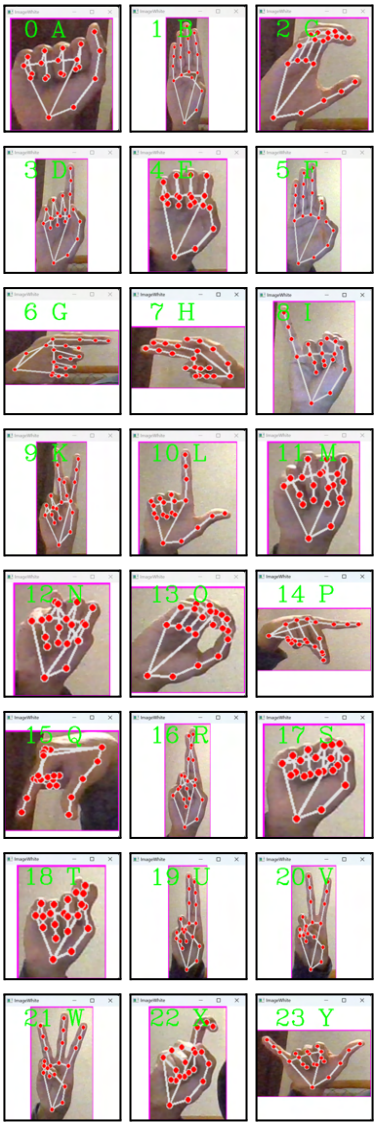
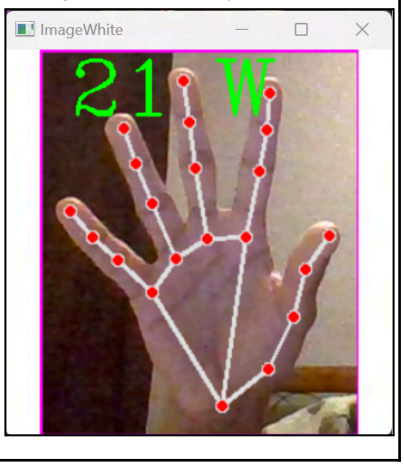
Boundary	Very low lighting conditions	Correct classification of hand sign.	✓ Correct classification of hand sign. 

Boundary	Hand sign at the edge of camera frame.	Do not detect hand.	×
			`Traceback (most recent call last):`
			`  File "C:\Users\chowd\PycharmProjects\Fingerspelling - NEA\main.py", line 37, in <module>`
			`  imgResize = cv2.resize(imgCrop, (wCal, imgSize))  # Resize imgCrop`
			`cv2.error: OpenCV(4.9.0) D:\a\opencv-python\opencv-python\opencv\modules\imgproc\src\resize.cpp:4152: error: (-215:Assertion failed) !ssize.empty() in function 'cv::resize'`

Invalid	Completely dark frame	Do nothing.	✓ Do nothing. 

Boundary	More than one hand detected in the same frame.	Continue sending image of first hand detected to the model.	✓ Continue sending image of first hand detected to the model. 
Invalid	Non-hand object in the frame.	Do not detect an object as a hand.	✓ Do not detect an object as a hand.
Normal	Hand sign with a slight blur motion.	Correct classification of hand sign.	✓ Correct classification of hand signs. 
Boundary	Fast-moving hand sign.	Do not detect hand.	✓ Do not detect hand. 

Invalid	Extremely blurred or obscured hand sign.	Do not detect hand.	✓ Do not detect hand. 
Normal	Different skin tones.	Detect hand.	✓ Detech hand. 
Normal	Hand close to camera.	Detect hand.	✓ Detect hand. 
Invalid	Hand very far away from camera.	Do not detect hand.	✓ Do not detect hand. 

Normal	Hand sign for each letter (except J and Z).	Correct classification of hand sign.	✓ Correct classification of hand sign. 
Boundary	Any other hand gesture that is not part of the ASL alphabet.	Classify sign based on which hand gesture it closely resembles.	✓ Classify sign based on which hand gesture it closely resembles. 

The recognition model passed most of the tests. Now I just need to draw an appropriate bounding box, remove the skeleton, and add the model's prediction on the top of the hand, as shown in my GUI design.

To do this, I am going to use OpenCV's function to put text on top of the `img`, and draw the necessary rectangles.

Also, the code gives the following error whenever either `x` or `y` coordinates of the current bounding box is negative, and therefore it can't process `imgWhite`.

```
Traceback (most recent call last):
  File "C:\Users\chowd\PycharmProjects\Fingerspelling - NEA\main.py", line 37, in
<module>
    imgResize = cv2.resize(imgCrop, (wCal, imgSize))  # Resize imgCrop
cv2.error: OpenCV(4.9.0)
D:\a\opencv-python\opencv-python\opencv\modules\imgproc\src\resize.cpp:4152: error:
(-215:Assertion failed) !ssize.empty() in function 'cv::resize'
```

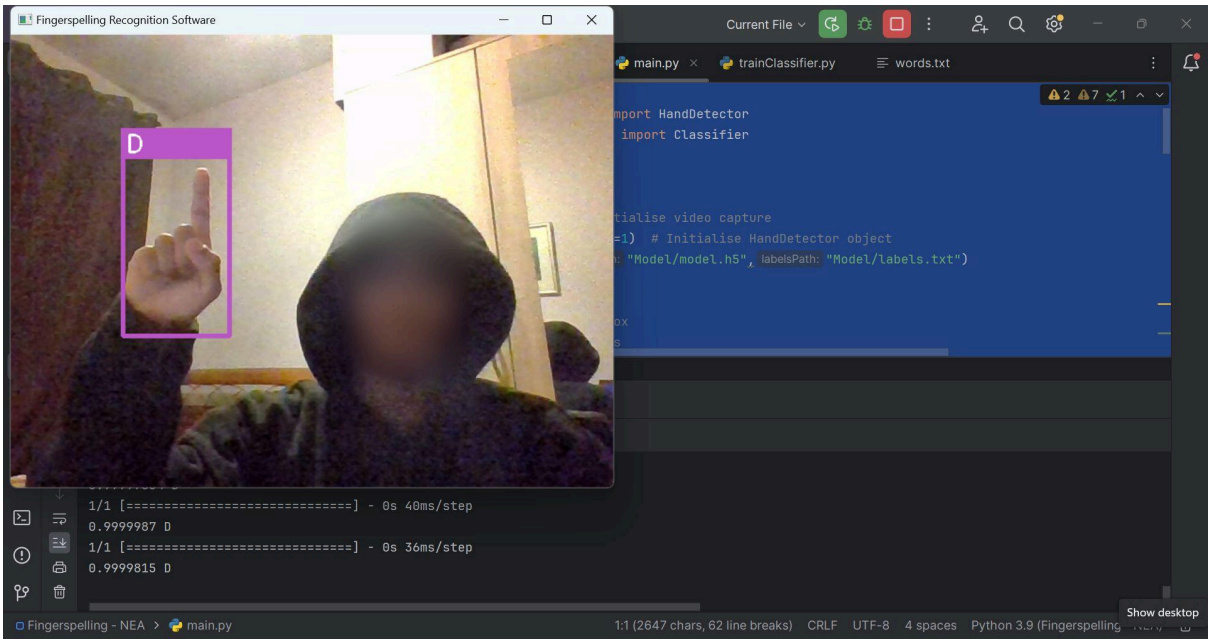
To fix this I can just use an if function, before resizing `imgCrop` into `imgWhite`.

```
import ...
...

# Constants
...

labels = ["A", "B", "C", "D", "E", "F", "G", "H", "I", "K", "L", "M", "N", "O",
"P", "Q", "R", "S", "T", "U", "V", "W", "X", "Y"]


while True:
   success, img = cap.read()  # Capture frame from camera
   imgOutput = img.copy()  # Main display
   hands, img = detector.findHands(img)  # Detect hands and annotate the frame

   if hands:
      ...

       if (x - offset) > 0 and (y - offset) > 0:  # Prevents error when x/y
coordinates are negative
       # Height is greater
           if aspectRatio > 1:
               ...
               prediction, index = classifier.getPrediction(imgWhite)

           # Width is greater
           else:
               ...
               prediction, index = classifier.getPrediction(imgWhite)

       print(prediction[index], labels[index])

       # Bounding box
       cv2.rectangle(imgOutput, (x - offset - 2, y - offset - 30), (x + w + 22,
y - offset), (201, 87, 188), cv2.FILLED)
       cv2.putText(imgOutput, labels[index], (x - 17, y - 23),
```

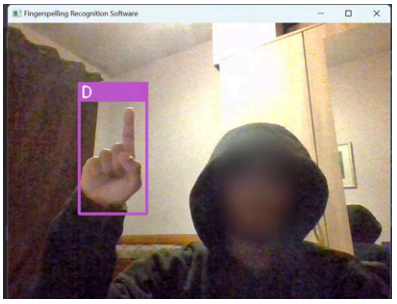
```
cv2.FONT_HERSHEY_PLAIN, 2, (255, 255, 255), 2)
       cv2.rectangle(imgOutput, (x - offset, y - offset), (x + w + offset, y + h
+ offset), (201, 87, 188), 3)

   cv2.imshow("Fingerspelling Recognition Software", imgOutput)
   cv2.waitKey(1)
```

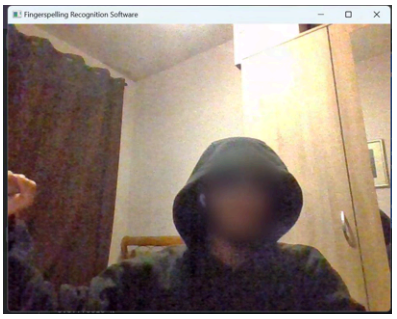
Here, I can't really remove the "skeleton" of the hand by simply changing `hands, img = detector.findHands(img)` into `hands = detector.findHands(img,draw=False)`, as I need it to be in `imgWhite`, so that it can be fed into the model to receive an accurate prediction. This is because the model was trained on images with the skeleton, therefore the input has to be the same format, if I want to receive a high accuracy prediction. Therefore I copied `img` into a new variable called `imgOutput` and displayed that instead of `img`. This is where I am going to draw the bounding box. This was also very simple, by drawing 1 box with top right coordinates slightly above original `x` and `y`, and adding the letter on top of it. Then draw the bounding box around the hand with the same colour.

Output:



Testing Iteration 1:

Test		Result	
Type	**Input**	**Expected**	**Output**
Normal	Hand detected in camera feed.	Draw a bounding box around hand, with model prediction on top.	✓ Draw a bounding box around hand, with model prediction on top. 

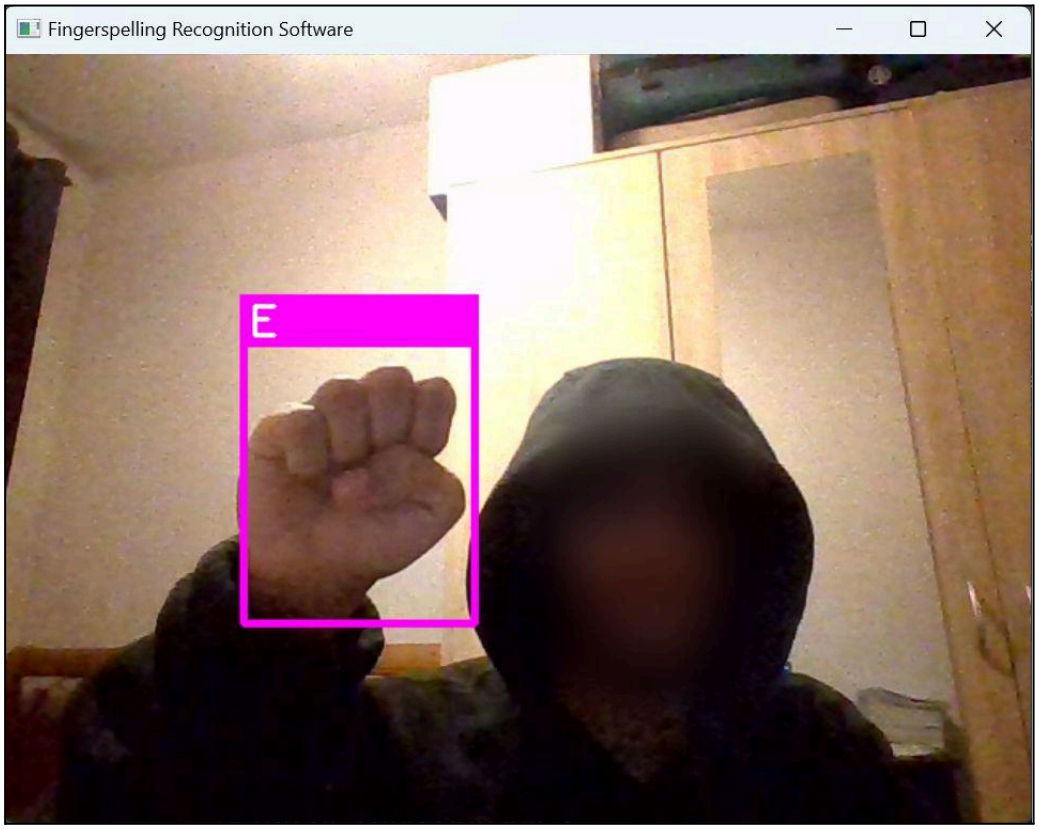
Boundary	Hand sign at the edge of camera frame.	Do not detect hand.	✓ Do not detect hand. 

Stakeholders Feedback

"What do you think of the fingerspelling recognition system?"

Yusuf *“Looks good! Don't you think that the colour of the bounding box is a bit too dark?”*

"Yes you are right, I will change it to a purple that is brighter."



“I changed the colour to a lighter pink colour (HEX: #FF00FF) □ so that it is more visible.”

"Yes, that is perfect. The hand is now easily detectable."

Robert *“Simple, it looks fine.”*

Imtiaz *“The bounding box looks nice and clean.”*

Iteration 1 Review:
In this part, I set up the model, testing different ways making sure everything works smoothly. Below are all the points from the success criteria that I have met:

Requirement	Sub-Requirement	User Requirement	Success Criteria
Recognition	Hand localisation	Bounding box	- Box around the hand will be shown in every frame. - Box will follow hand movements. - Box will be x pixels away from the main joints of the hand. - One hand will be detected at a time, (software will not place a bounding box around another hand, if one hand is already being detected).
Real-Time Feedback	Output	Letter prediction	- Display model prediction in the recognition of the sign (letter). - Display letter on top of the bounding box, on the left. - Letter will change based on the sign. - Letter should be a different colour than the bounding box.
Functionality	Stop Software	X button	- The X button will be in the top left corner of the window. - When the mouse hovers over the X button, it will turn red. - If pressed, the window will close.

Specification points that I had to change is:

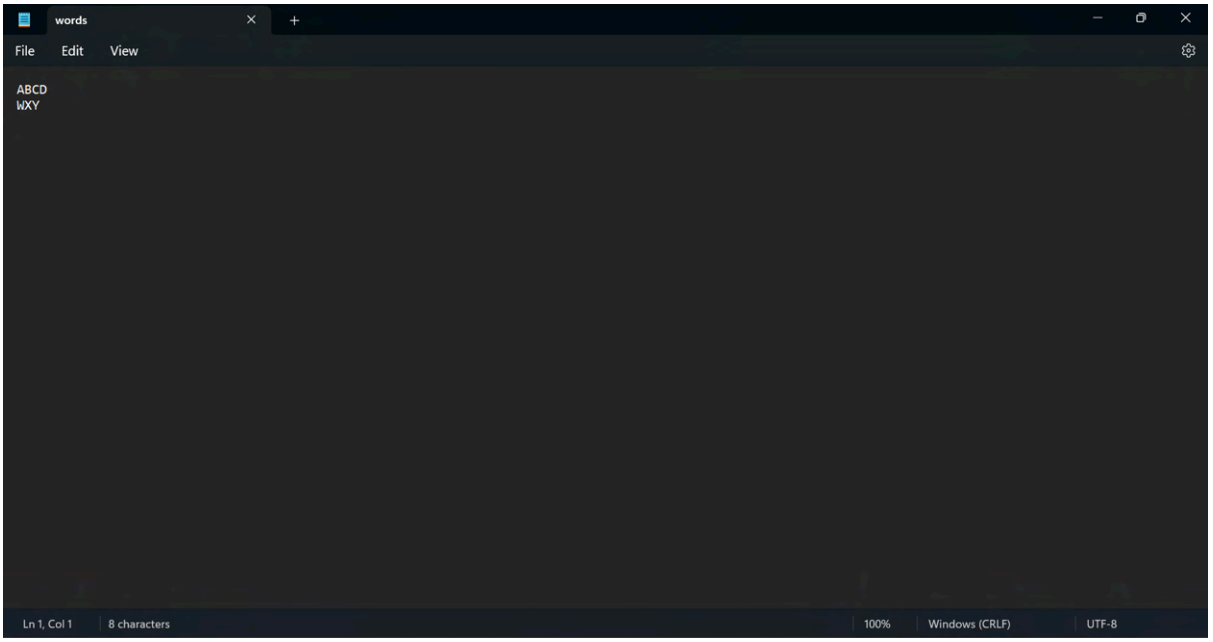
Requirement	Sub-Requirement	User Requirement	Success Criteria

Functionality	GUI colours	Bounding box colour	- Colour of the bounding box should be a bright colour:
			  - HEX: #C957BC
			  - RGB: rgb(201, 87, 188)

This is because after stakeholder feedback, it is a bit darker than expected, and therefore stakeholders may find it difficult to see if the hand is detected by the software.

2nd Iteration - Learning Environment

For this section, I plan to follow the flow chart that I created in the Design section. Instead of storing all the words in a list, I want to store them in a text file, where the code will retrieve them and put every word in a list.



This list will be called `words.txt` and will only contain the words `ABCD` and `XYW` to test the following code.

```
import ...

...

# Constants
...

# Words Constants
labels = ["A", "B", "C", "D", "E", "F", "G", "H", "I", "K", "L", "M", "N", "O",
"P", "Q", "R", "S", "T", "U", "V", "W",
         "X", "Y"]
wordsFile = "words.txt"
currentWordIndex = 0
currentLetterIndex = 0


def get_prediction(imgWhite):
   modelPrediction, modelIndex = classifier.getPrediction(imgWhite)
   return modelPrediction, modelIndex


def words_list(wordsFile):
   wordsList = []
   with open(wordsFile) as file:
       for line in file:
```

```
            wordsList.append(line.strip())
    return wordsList


while True:
  ...

    if hands:
        ...

        # ASL Alphabet Learning
        letter = labels[index]  # Model letter prediction
        words = words_list(wordsFile)  # Place all words into a list
        currentWord = list(words[currentWordIndex])  # Split currentWord into
individual letters
        currentLetter = currentWord[currentLetterIndex]

        print(words[currentWordIndex], currentWordIndex)
        print(letter, currentLetter, currentLetterIndex)

        if currentLetter == letter:  # Check if model's prediction matches with
currentLetter
            if currentLetterIndex >= (len(currentWord) - 1):  # Check if
currentLetter reaches end of currentWord
                currentWordIndex += 1  # Move onto next word in the list
                currentLetterIndex = 0  # First letter of new word
            else:
                currentLetterIndex += 1  # Move onto next letter in currenWord

    cv2.imshow("Fingerspelling Recognition Software", imgOutput)
    cv2.waitKey(1)
```

Each word in this list is broken down into individual letters, and the system challenges the user to finger-spell these letters in sequence.

Initially, the `words_list` function reads all the words from `words.txt` and stores them in a list. When a hand is detected, the model predicts which `letter` is being signed. This predicted `letter` is then matched against the current letter the user is supposed to spell, based on the current word and letter indices (`currentWordIndex` and `currentLetterIndex`). If the prediction matches the intended letter, the system moves to the next letter in the word (`currentLetterIndex += 1`). Once all letters of the current word are correctly finger-spelt, the system advances to the next word in the list (`currentWordIndex += 1`), resetting `currentLetterIndex` to `0` to start spelling the new word from its first letter.

Output:

```
1/1 [==============================] - 1s 1s/step
ABCD 0
C A 0
1/1 [==============================] - 0s 29ms/step
ABCD 0
A A 0
1/1 [==============================] - 0s 30ms/step
ABCD 0
```

```
A B 1
1/1 [==============================] - 0s 42ms/step
ABCD 0
A B 1
1/1 [==============================] - 0s 27ms/step
ABCD 0
G B 1
1/1 [==============================] - 0s 46ms/step
ABCD 0
E B 1
1/1 [==============================] - 0s 31ms/step
ABCD 0
F B 1
1/1 [==============================] - 0s 38ms/step
ABCD 0
G B 1
1/1 [==============================] - 0s 30ms/step
ABCD 0
G B 1
1/1 [==============================] - 0s 30ms/step
ABCD 0
F B 1
1/1 [==============================] - 0s 44ms/step
ABCD 0
B B 1
1/1 [==============================] - 0s 28ms/step
ABCD 0
B C 2
1/1 [==============================] - 0s 30ms/step
ABCD 0
G C 2
1/1 [==============================] - 0s 35ms/step
ABCD 0
B C 2
1/1 [==============================] - 0s 28ms/step
ABCD 0
B C 2
1/1 [==============================] - 0s 27ms/step
ABCD 0
F C 2
1/1 [==============================] - 0s 29ms/step
ABCD 0
C C 2
1/1 [==============================] - 0s 40ms/step
ABCD 0
C D 3
1/1 [==============================] - 0s 29ms/step
ABCD 0
C D 3
1/1 [==============================] - 0s 28ms/step
ABCD 0
C D 3
1/1 [==============================] - 0s 42ms/step
ABCD 0
C D 3
1/1 [==============================] - 0s 28ms/step
```

```
ABCD 0
C D 3
1/1 [==============================] - 0s 26ms/step
ABCD 0
C D 3
1/1 [==============================] - 0s 43ms/step
ABCD 0
C D 3
1/1 [==============================] - 0s 33ms/step
ABCD 0
D D 3
Traceback (most recent call last):
  File "C:\Users\chowd\PycharmProjects\NEA Original\test.py", line 84, in <module>
    currentWord = list(words[currentWordIndex])  # Split currentWord into individual
letters
IndexError: list index out of range
1/1 [==============================] - 0s 38ms/step
```

The output of the code was tested with only `ABCD` in `words.txt`. The code works perfectly fine, however as `currentWordIndex` reaches the end of the list it gives the error at the end. This is because I didn't set any condition to happen whenever the index reaches the end of the list.

Testing Iteration 2:

Test		Result	
Type	**Input**	**Expected**	**Output**

Normal	Letter prediction given by the model matches with the letter of the word.	Move onto next letter.	✓ Move onto next letter.
			`1/1 [==============================] - 1s 1s/step`
			`ABCD 0`
			`C A 0`
			`1/1 [==============================] - 0s 29ms/step`
			`ABCD 0`
			`A A 0`
			`1/1 [==============================] - 0s 30ms/step`
			`ABCD 0`
			`A B 1`

Invalid	Letter prediction given by the model does not match with the letter of the word.	Do nothing.	✓ Do nothing.
			`1/1 [==============================] - 0s 42ms/step`
			`ABCD 0`
			`A B 1`
			`1/1 [==============================] - 0s 27ms/step`
			`ABCD 0`
			`G B 1`

Boundary	Letter prediction given by the model does not match with the **last** letter of the word.	Move onto next word.	✓ Move onto next word.
			`1/1 [==============================] - 0s 28ms/step`
			`ABCD 0`
			`O D 3`
			`1/1 [==============================] - 0s 41ms/step`
			`ABCD 0`
			`D D 3`
			`1/1 [==============================] - 0s 28ms/step`
			`WXY 1`
			`D W 0`

Boundary	Word is last word on the list	Stop programming.	✕
			`Traceback (most recent call last):`
			`File "C:\Users\chowd\PycharmProjects\NEA Original\test.py", line 84, in <module>`
			`currentWord = list(words[currentWordIndex]) # Split currentWord into individual letters`
			`IndexError: list index out of range`

Stakeholders Feedback	
"What do you think of the fingerspelling recognition system?"	
Yusuf	*"I like it."*
Robert	*"It looks fine. Well done! You could change the code so that if all the words are fingerspelled, reset the list and go back to the first word."* *"Great idea! I will implement it right now."*
Imtiaz	*"Perfect! I am surprised at how smoothly it works."*

Now I am going to apply my stakeholder's feedback as it is a better idea rather than just stopping the program. To do this, I can just simply use an if statement, checking if `currentWordIndex` is bigger than the length of `words`, and resetting the index to `0`.

```
# ASL Alphabet Learning
letter = labels[index]  # Model letter prediction
words = words_list(wordsFile)  # Place all words into a list
currentWord = list(words[currentWordIndex])  # Split currentWord into individual letters
currentLetter = currentWord[currentLetterIndex]
```

```
print(words[currentWordIndex], currentWordIndex)
print(letter, currentLetter, currentLetterIndex)

if currentLetter == letter:  # Check if model's prediction matches with
currentLetter
   if currentLetterIndex >= (len(currentWord) - 1):  # Check if currentLetter
reaches end of currentWord
       currentWordIndex += 1  # Move onto next word in the list
       currentLetterIndex = 0  # First letter of new word
       if currentWordIndex == len(words):  # Check if there are no words left
           currentWordIndex = 0  # Go back to initial word
   else:
       currentLetterIndex += 1  # Move onto next letter in currentWord
```

Testing Iteration 2:

Test		Result	
Type	**Input**	**Expected**	**Output**
Boundary	Word is last word on the list	Loop back to first word.	✓ Loop back to first word. 1/1 [==========] ==] - 0s 42ms/step WXY 1 G Y 2 1/1 [==========] ==] - 0s 28ms/step WXY 1 Y Y 2 1/1 [==========] ==] - 0s 29ms/step ABCD 0 Y A 0

Stakeholders Feedback	
"What do you think of the fingerspelling recognition system?"	
Yusuf	*“That's better, as it allows for continuous practice.”*
Robert	*“Very nice.”*
Imtiaz	*“Yes, that change is very useful.”*

Iteration 2 Review:
The program successfully checks if the user sign matches with the letter given. If it does, it moves onto the next letter. If it does then it moves onto the next letter, and so on, until the whole word is completely fingerspelled. If the user fingerspelled all the words in the list, software loops back to the first word in the list.

Below are the points that the code meets from the success criteria:

Requirement	Sub-Requirement	User Requirement	Success Criteria
Functionality	Interactive Learning Environment	Letter	- If the user's sign matches with the software's given letter, move onto the next letter.
		Word	- If the whole word is fingerspelled, move onto the next word on the list.

Now, I can move onto displaying the following result in front of the camera.

3rd Iteration - Graphical User Interface (GUI)

Before I move onto developing the other features of my software, I want to construct the GUI so that I can display the output clearly to visualise. To do this I am going to use OpenCV's module to draw the necessary parts, just like I did when drawing the bounding box. For the GUI I will follow the initial design that I made.

```
import ...

...

# Constants
...

# Words Constants
...

# Functions
...

def end_of_words_check(currentWordIndex, words):
    if currentWordIndex == len(words):  # Check if there are no words left
        currentWordIndex = 0  # Go back to initial word
    return currentWordIndex


# Main
while True:
    ...

    if hands:
        ...

    # GUI
    cv2.rectangle(imgOutput, (400, 0), (1000, 1000), (179, 227, 255), cv2.FILLED)
# Background
    cv2.putText(imgOutput, currentLetter, (485, 200), cv2.FONT_HERSHEY_PLAIN, 7,
(146, 32, 117), 7)  # currentLetter
    cv2.putText(imgOutput, words[currentWordIndex], (419, 270),
cv2.FONT_HERSHEY_PLAIN, 4, (146, 32, 117), 4)  # currentWord
    cv2.putText(imgOutput, "Next word: " +
words[end_of_words_check(currentWordIndex+1, words)], (442, 300),
cv2.FONT_HERSHEY_PLAIN, 1, (146, 32, 117), 1)  # Next word

    cv2.imshow("Fingerspelling Recognition Software", imgOutput)
    cv2.waitKey(1)
```

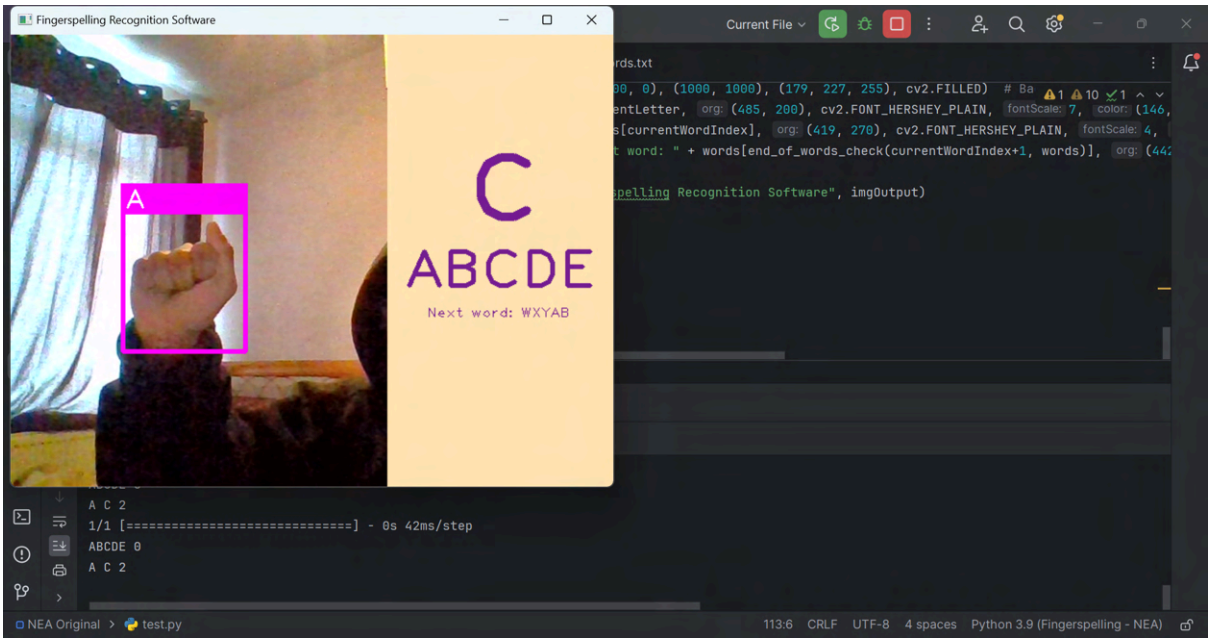
The `cv2.rectangle` function is used to create a filled rectangle, serving as a background for text information. This visual element improves readability and aesthetics. The rectangle's dimensions and colour are defined to ensure clear visibility of the text against the GUI background.

The `cv2.putText` function displays three key pieces of information: the current letter being identified by the model (`currentLetter`), the full word currently being spelt out (`words[currentWordIndex]`), and the next

word in the learning sequence (`"Next word: " + words[end_of_words_check(currentWordIndex+1, words)]`). The size, colour, and position of the text are meticulously chosen to facilitate easy reading and engagement from the user. This approach not only aids in learning but also adds a structured flow to the educational content, making it easier for users to follow along and understand their progress through the material.

Furthermore, the `end_of_words_check` function ensures that the sequence of words loops back to the beginning once the end of the list is reached, enabling continuous learning without manual resets. This automatic transition to the next word, along with visual cues for the current and upcoming words, creates an interactive and immersive learning experience for users, aiding in the effective memorisation and practise of ASL alphabet and vocabulary.

Output:



I tested the code with the words `ABCDE` and `WXYAB` in `words.txt`.I have decided to position the next word below the `currentWord`, as it does not fit next to it. OpenCV does not have a wide range of options when it comes to sizing/fonts. The design still looks perfectly similar.

Since this section is entirely about building the GUI, it does not have any testing to do.

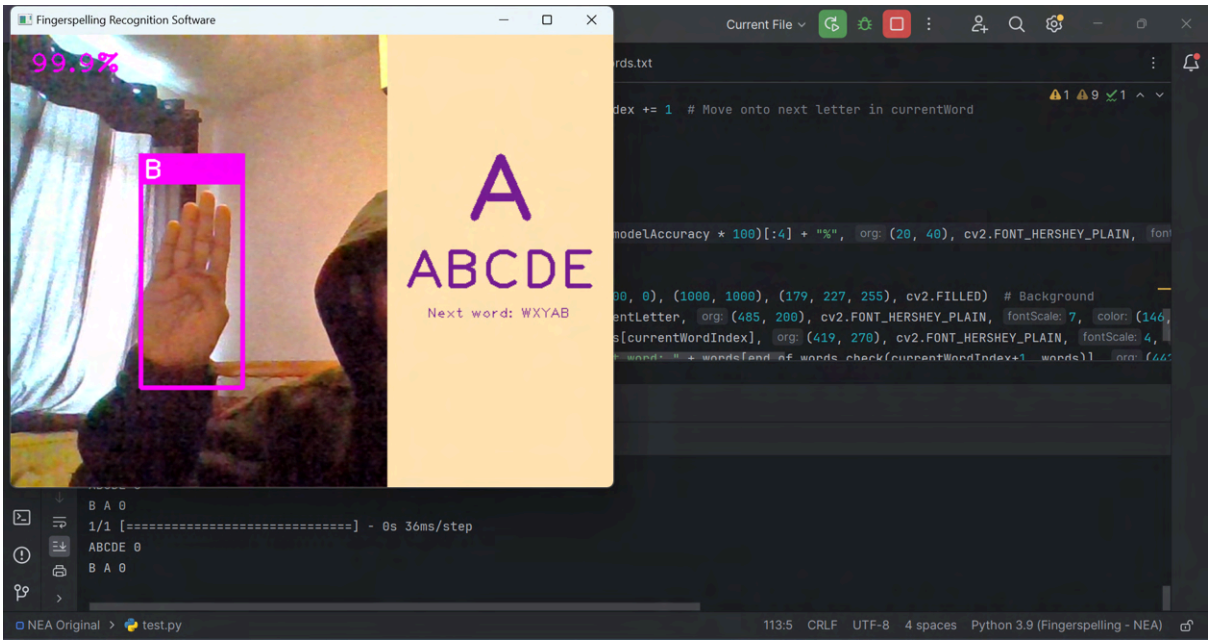
Stakeholders Feedback	
"What do you think of the GUI? I can't position the current word and the next word next to each other as shown in the initial GUI design, so I have decided to put it underneath."	
Yusuf	*"I think that is fine. I don't mind the change"*
Robert	*"Very clean design. Well done."*
Imtiaz	*"That is better than the original design. I think you forgot to add the accuracy of the model as a percentage as you mentioned."* *"Yes, I will do that right now."*

```
import ...

...

# Constants
...

# Words
...

modelAccuracy = 0

# Functions
...

while True:
    ...

    if hands:
        ...

        if (x - offset) > 0 and (y - offset) > 0:  # Prevents error when x/y
coordinates are negative
            ...

             modelAccuracy = prediction[index]
        ...

    else:
        modelAccuracy = 0

    # Model Accuracy
    cv2.putText(imgOutput, str(modelAccuracy * 100)[:4] + "%", (20, 40),
cv2.FONT_HERSHEY_PLAIN, 2, (255, 0, 255), 2)

    cv2.imshow("Fingerspelling Recognition Software", imgOutput)
    cv2.waitKey(1)
```

The `modelAccuracy` variable is central to this functionality. It is initialised to 0 outside the main loop to ensure it has a defined value before any hands are detected. Within the loop, if the software detects a hand (`if hands:`) and confirms that the detected hand's bounding box coordinates (`x`, `y`) are within the frame boundaries (preventing errors due to negative coordinates), modelAccuracy is updated to reflect the current prediction accuracy obtained from the classifier for the detected hand gesture. If no hands are detected, `modelAccuracy` is reset to `0` to indicate the absence of predictions.

The critical piece of user feedback is provided via `cv2.putText`, which overlays the model's accuracy percentage on the `imgOutput` image. This information is formatted as a string, displaying only the first four characters (to avoid overly long numbers) followed by a percentage sign, ensuring the accuracy value is easily understandable. The accuracy percentage is positioned at coordinates (20, 40) on the display, using a distinct colour (magenta, in BGR colour space as (`255, 0, 255`)) and a noticeable font size for clear visibility.

Output:



Iteration 3 Review:

The GUI design looks the way me and my stakeholders expected. Here are all the points that I have met from the success criteria:

Requirement	Sub-Requirement	User Requirement	Success Criteria
Recognition	Camera access	Access user's front camera	- On the camera window, the right hand side will be covered with the GUI, and on the left hand side the user will be able to see his hand.
Real-Time Feedback	Output	Model accuracy	- Display model accuracy as a percentage. - Display accuracy on the top left corner of the window. - Accuracy of the model will be medium sized.
Functionality	Interactive Learning Environment	Letter	- Letter will be displayed in the centre of the right hand side of the window. - Size of the letter will be very big.
		Word	- Word will be displayed underneath the letter to be fingerspelled. - Next word to be fingerspelled is also displayed next to the current word.

	GUI colours	GUI background	- Background for the letters, words and instructions should be present on the left half of the screen.
			- Colour of the background should light:
			  - HEX: #FFE3B3
			  - RGB: rgb(255, 227, 179)

		Text colours	- Colour of the text should be dark:
			  - HEX: #752092
			  - RGB: rgb(117, 32, 146)

4th Iteration - Skip Letter/Word

This is simple to build, as I can use the concept that I used when collecting data in `dataCollection.py`. This means that I can just wait for a key to be pressed and if the key is pressed, do skip `currentLetter` or `currentWord`.

```
key = cv2.waitKey(1)

# Skip Letter
if key == ord("l"):
    print(currentLetter, "is skipped.")
    currentLetterIndex += 1  # Increment Letter

# Skip Word
if key == ord("w"):
    print(words[currentWordIndex], "is skipped.")
    currentWordIndex += 1  # Increment Word
```

This code introduces keyboard shortcuts to enhance user interaction within a fingerspelling recognition application, enabling users to skip either the current letter or the entire word during practice. By pressing (`L`), users can advance to the next letter within the word, while pressing (`W`) allows them to move to the next word in the sequence. This functionality is achieved by capturing keyboard input through `cv2.waitKey(1)` and adjusting the `currentLetterIndex` or `currentWordIndex` accordingly. Feedback is provided to the user via console messages, indicating the letter or word being skipped.

The code is tested with the words `ABCDE` and `WXYAB` in `words.txt`.

Testing Iteration 4:

Test		Result	
Type	**Input**	**Expected**	**Output**

Normal	Press (`L`) key.	Move onto next letter in the word.	✓ Move onto the next letter in the word.
			`A is skipped.`
			`B is skipped.`

Boundary	Press (`L`) key and `currentLetter` is the last letter of `currentWord`.	Move onto the first letter in the next word.	×
			`A is skipped.`
			`B is skipped.`
			`C is skipped.`
			`D is skipped.`
			`E is skipped.`
			`Traceback (most recent call last):`
			`File "C:\Users\chowd\PycharmProjects\NEA Original\test.py", line 52, in <module>`
			`currentLetter = currentWord[currentLetterIndex]`
			`IndexError: list index out of range`

Normal	Press (`W`) key.	Move onto the next word in the list of words.	✓ Move onto the next word in the list of words.
			`ABCDE is skipped.`
			`WXYAB is skipped.`

Boundary	Press (`W`) key and `currentWord` is the last word in `words`.	Move onto the first word in the list.	×
			`ABCDE is skipped.`
			`WXYAB is skipped.`
			`Traceback (most recent call last):`
			`File "C:\Users\chowd\PycharmProjects\NEA Original\test.py", line 51, in <module>`
			`currentWord = list(words[currentWordIndex]) # Split currentWord into individual letters`
			`IndexError: list index out of range`

Invalid	Press any other key.	Do nothing.	✓ Do nothing.

I did not consider the boundary situations of my code. The rest works perfectly.

```
key = cv2.waitKey(1)

# Skip Letter
if key == ord("l"):
    print(currentLetter, "is skipped.")
    if currentLetterIndex >= (len(currentWord) - 1):  # Check if currentLetter reaches end of currentWord
        currentWordIndex += 1  # Move onto next word in the list
        currentLetterIndex = 0  # First letter of new word
        currentWordIndex = last_word(currentWordIndex, words)
    else:
        currentLetterIndex += 1  # Increment Letter

# Skip Word
if key == ord("w"):
    print(words[currentWordIndex], "is skipped.")
    currentWordIndex = last_word(currentWordIndex+1, words)
```

Here, I just copied part of the code to check if `currentLetter` reaches the end of `currentWord`. When the user presses (`L`) to skip a letter, the program now checks whether the current letter is the last in the word. If it is, the program advances to the first letter of the next word, effectively moving to a new word once all letters have been skipped. This transition is facilitated by the `last_word` (changed name of `end_of_words_check` to `last_word`) function, which likely checks and adjusts the currentWordIndex to loop back to the first word if the end of the word list is reached, ensuring continuous practice without interruption. Similarly, pressing (`W`) skips the current word entirely, with the `last_word` function again ensuring the index is adjusted correctly for a cyclic word list traversal.

Testing Iteration 4:

Test		Result	
Type	**Input**	**Expected**	**Output**

Boundary	Press (`L`) key and `currentLetter` is the last letter of `currentWord`.	Move onto the first letter in the next word.	✓ Move onto the first letter in the next word.
			`A is skipped.`
			`B is skipped.`
			`C is skipped.`
			`D is skipped.`
			`E is skipped.`
			`W is skipped.`
			`X is skipped.`
			`Y is skipped.`
			`A is skipped.`
			`B is skipped.`
			`A is skipped.`
			`B is skipped.`
			`C is skipped.`
			`D is skipped.`
			`E is skipped.`

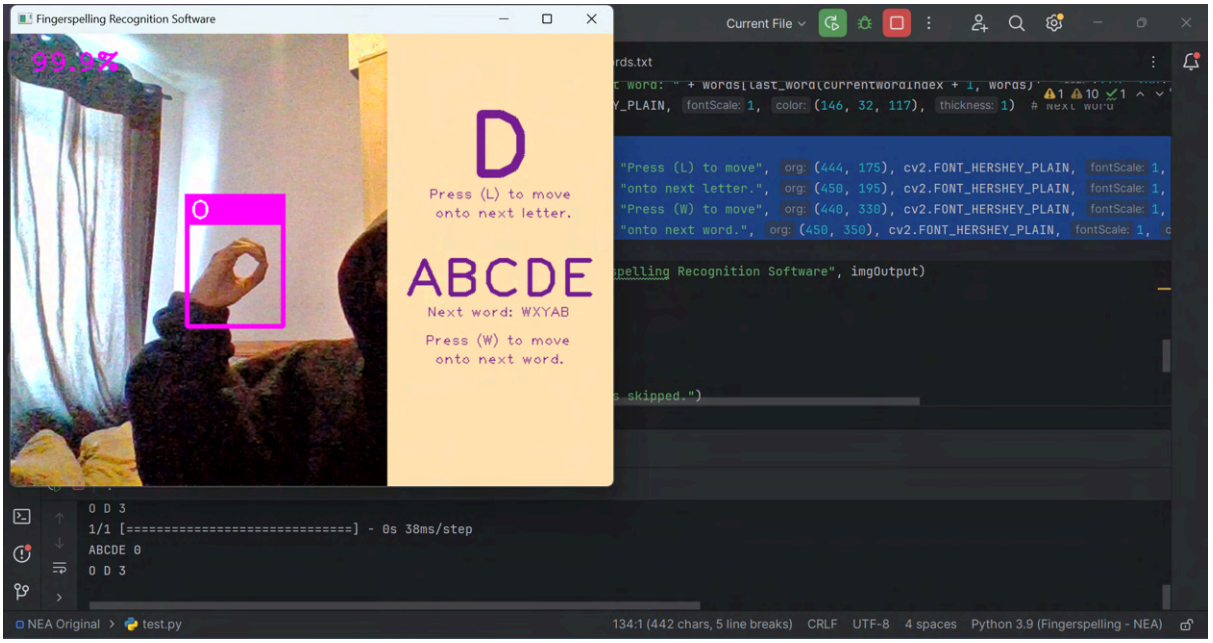
Boundary	Press (`W`) key and `currentWord` is the last word in `words`.	Move onto the first word in the list.	✓ Move onto the first word in the list.
			`ABCDE is skipped.`
			`WXYAB is skipped.`
			`ABCDE is skipped.`

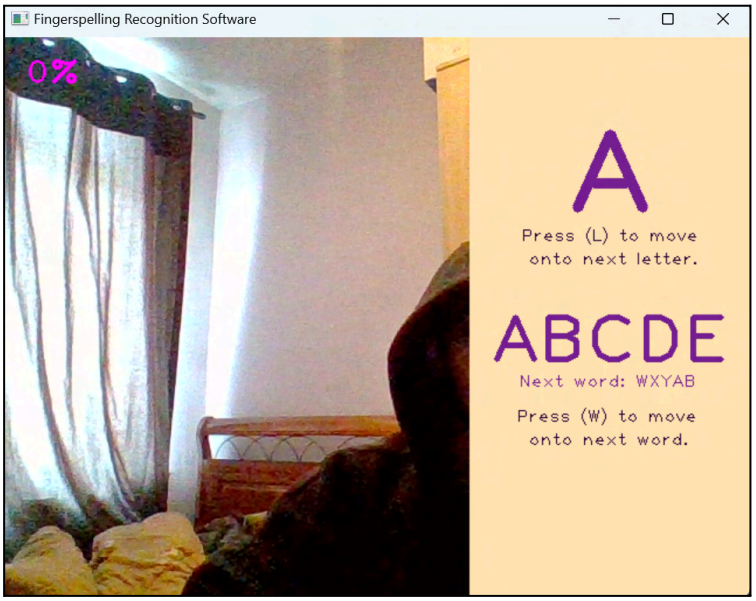
The currentLetter and currentWord in the GUI changes accordingly. Now I can just use OpenCV to write the instructions on the screen.

```
# Skip Instructions
cv2.putText(imgOutput, "Press (L) to move", (444, 175), cv2.FONT_HERSHEY_PLAIN,
1, (146, 32, 117), 1)
cv2.putText(imgOutput, "onto next letter.", (450, 195), cv2.FONT_HERSHEY_PLAIN,
1, (146, 32, 117), 1)
cv2.putText(imgOutput, "Press (W) to move", (440, 330), cv2.FONT_HERSHEY_PLAIN,
1, (146, 32, 117), 1)
cv2.putText(imgOutput, "onto next word.", (450, 350), cv2.FONT_HERSHEY_PLAIN, 1,
(146, 32, 117), 1)
```

For this, since the font size can't go smaller than 1, I had to make some changes. The line "Press (L) to move onto the next letter." had to be written in 2 separate lines. Same with "Press (W) to move onto the next word." The is positioned slightly below the current letter to be fingerspelled and "Press (W) to move onto the next word." is positioned below the next word.

Output:



Stakeholders Feedback	
"I have made some changes to the GUI, so that everything fits together. What do you think of these changes?"	
Yusuf	*"I like it, but can you change the text colour of the instruction, so that it is more distinguishable from the main components of the program."* *"I will change that."*  *"I made it darker. What do you think?"* *"Perfect."*
Robert	*"Better than the initial GUI design that you showed me."*
Imtiaz	*"I think it is ok. Not a major difference from the initial design."*

Iteration 4 Review:
In this iteration, I managed to successfully give my stakeholders the possibility of skipping the letter or word. This is very important as it may be that my stakeholders already know how to ho fingerspell, or the model does not recognise the sign correctly, giving wrong letter prediction.

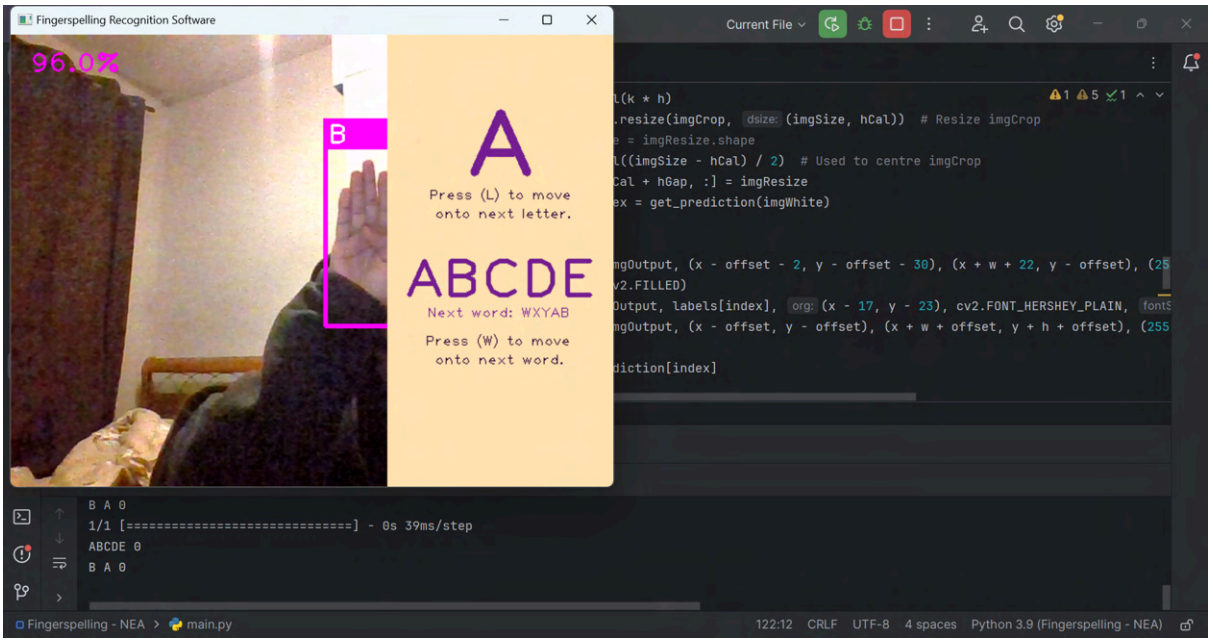
Below are all the points that I have met in my success criteria:

Requirement	Sub-Requirement	User Requirement	Success Criteria
Functionality	Interactive Learning Environment	Skip letter function	- Users should be able to skip to the next letter, with the press of a button (L). - Display message *"Press (L) to move onto next letter."* underneath the letter. - Message will be underneath the current letter. - Message font size will be very small.
		Skip word function	- Users should be able to skip to the next word, with the press of a button (W). - Display next word, to let user know what word the user is going to fingerspell when he pressed the button. - Display message *"Press (W) to move onto next letter."* underneath the letter. - Message should be around a box. - Message will be underneath the current word. - Message font size will be very small.

5th Iteration - Real Time Feedback

To begin, I want to first adjust the code so that the hand bounding box is not created whenever the hand is fully or partially behind the GUI. This problem can be seen in the image below:

Output:



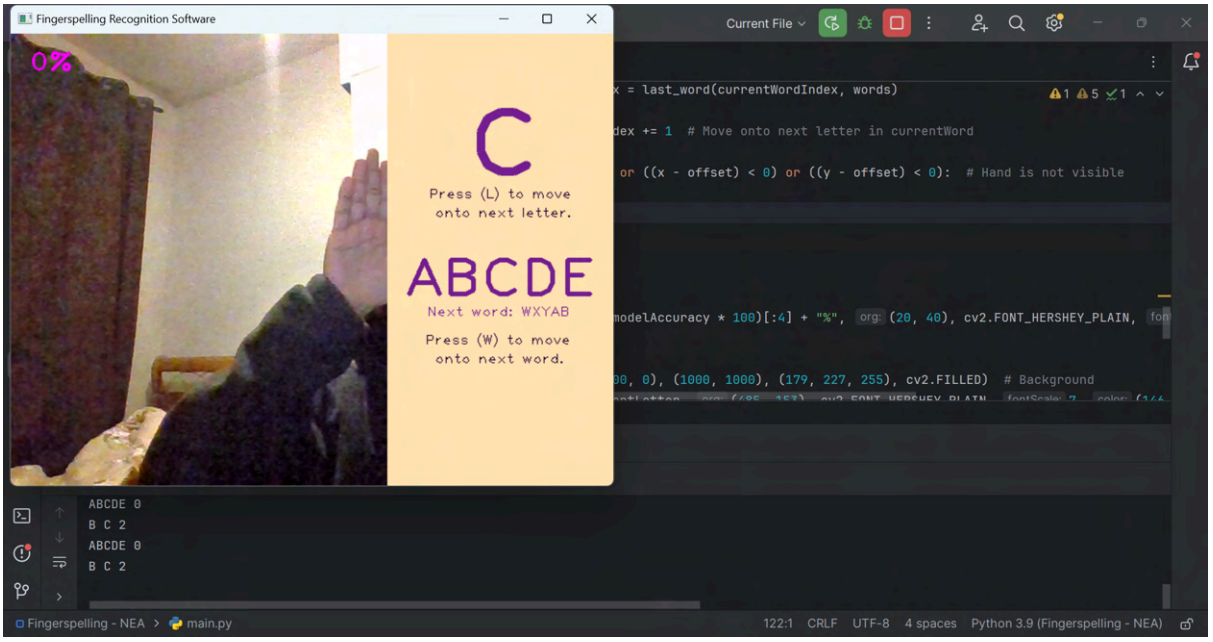
To address this issue, I will have to change when to draw the bounding box. This means that I am gonna have to change `modelAccuracy`, so that whenever it is behind the GUI, it is set to `0`.

```
import ...

...

# Constants
...

# Words
...

# Functions
...


while True:
    ...

    if hands:
        ...

        if (0 < (x - offset) < 320) and ((y - offset) > 0):  # Prevents error when x/y
coordinates are negative

            # Height is greater
            if aspectRatio > 1:
                ...
```

```
                prediction, index = get_prediction(imgWhite)

                # Bounding box
                cv2.rectangle(imgOutput, (x - offset - 2, y - offset - 30), (x + w + 22,
y - offset), (255, 0, 255),
                              cv2.FILLED)
                cv2.putText(imgOutput, labels[index], (x - 17, y - 23),
cv2.FONT_HERSHEY_PLAIN, 2, (255, 255, 255), 2)
                cv2.rectangle(imgOutput, (x - offset, y - offset), (x + w + offset, y + h
+ offset), (255, 0, 255), 3)

            # Width is greater
            else:
                ...
                prediction, index = get_prediction(imgWhite)

                # Bounding box
                cv2.rectangle(imgOutput, (x - offset - 2, y - offset - 30), (x + w + 22,
y - offset), (255, 0, 255),
                              cv2.FILLED)
                cv2.putText(imgOutput, labels[index], (x - 17, y - 23),
cv2.FONT_HERSHEY_PLAIN, 2, (255, 255, 255), 2)
                cv2.rectangle(imgOutput, (x - offset, y - offset), (x + w + offset, y + h
+ offset), (255, 0, 255), 3)

            modelAccuracy = prediction[index]

        ...

        if ((x - offset) > 320) or ((x - offset) < 0) or ((y - offset) < 0):  # Hand is
not visible
            modelAccuracy = 0

    else:
        modelAccuracy = 0

    # Model Accuracy
    cv2.putText(imgOutput, str(modelAccuracy * 100)[:4] + "%", (20, 40),
cv2.FONT_HERSHEY_PLAIN, 2, (255, 0, 255), 2)

    ...
```

The condition `if (x - offset) > 0 and (y - offset) > 0:` was changed to `if (0 < (x - offset) < 320) and ((y - offset) > 0):` so x coordinates are restricted between the GUI and the left hand side of the window. Then the bounding box is drawn if this condition is satisfied. This condition is also applied for modelAccuracy, setting it to 0, whenever the hand is not visible on the screen.

Output:



This now works perfectly. Now I can add the relevant messages. However there is a problem, since all the relevant information to display are dependent on the variables declared before drawing the GUI, if I use `cv2.putText`, the test will be beneath the background created, if I write the text before I constructed the GUI. Therefore I am going to have to access the information in `hands` again, after I draw the background.

The messages to be displayed are:

- "Place hand in front of camera to begin." - when the model hasn't been loaded yet.
- "Place hand in front of camera." - when the model has been loaded, and no hand is detected.
- "Move hand on the left hand side of the screen." - when hand is detected behind the background of the GUI
- "Well Done!" or "Correct!" - when user fingerspells letter correctly.

```
# Instructions
if len(hands) == 0 and instructionSetConstant == 0:  # Display initial message
   cv2.putText(imgOutput, "Place hand in front of", (425, 415),
cv2.FONT_HERSHEY_PLAIN, 1, (0, 0, 255), 1)
   cv2.putText(imgOutput, "camera to begin.", (450, 435),
cv2.FONT_HERSHEY_PLAIN, 1, (0, 0, 255), 1)

elif len(hands) == 0:  # Display message if no hands are detected
   cv2.putText(imgOutput, "Place hand in front", (434, 415),
cv2.FONT_HERSHEY_PLAIN, 1, (0, 0, 255), 1)
   cv2.putText(imgOutput, "of camera.", (475, 435), cv2.FONT_HERSHEY_PLAIN, 1,
(0, 0, 255), 1)

elif hands:

   instructionSetConstant = 1  # Change instructionSetConstant so that initial
message can no longer be displayed

   hand = hands[0]
   x, y, w, h = hand['bbox']  # Getting bounding box information
```

```
   if (x - offset) > 320:  # Inform user to move hand to the left
       cv2.putText(imgOutput, "Move hand on the left", (425, 415),
cv2.FONT_HERSHEY_PLAIN, 1, (0, 0, 255), 1)
       cv2.putText(imgOutput, "hand side of the screen.", (415, 435),
cv2.FONT_HERSHEY_PLAIN, 1, (0, 0, 255), 1)
   prediction, index = get_prediction(imgWhite)

   if currentLetter == labels[index]:

       randomMessage = random.randint(0, 1)  # Display random message

       if randomMessage == 1:
           cv2.putText(imgOutput, "Well Done!", (477, 425),
cv2.FONT_HERSHEY_PLAIN, 1, (0, 255, 0), 1)
       else:
           cv2.putText(imgOutput, "Correct!", (480, 425),
cv2.FONT_HERSHEY_PLAIN, 1, (0, 255, 0), 1)
```

When no hands are detected (`len(hands) == 0`), the application displays an initial message urging the user to place their hand in front of the camera to commence interaction. This message is adapted based on the `instructionSetConstant` variable, which ensures that the initial instructions are only shown once, shifting to a shorter prompt if the user's hand leaves the camera's view thereafter.

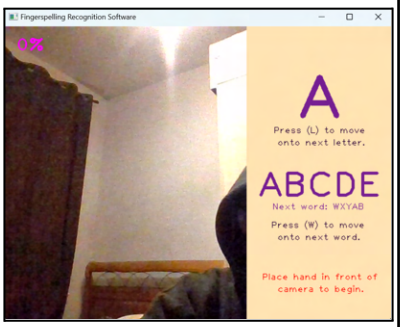
Upon detecting a hand (`elif hands:`), the `instructionSetConstant` is set to `1` to prevent the initial, longer instructional message from reappearing.

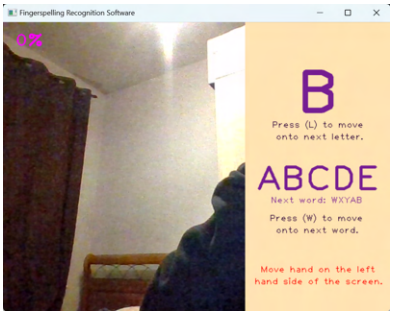
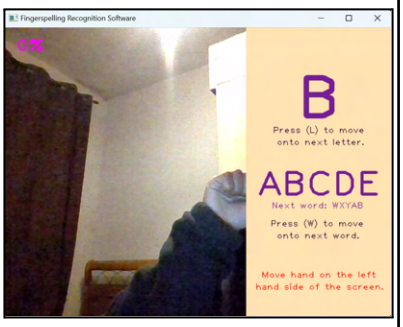
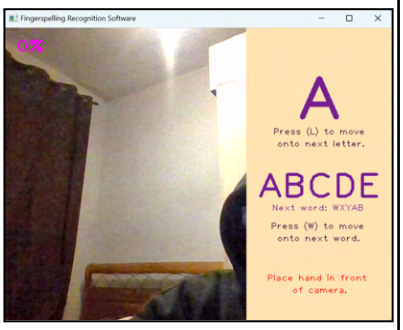
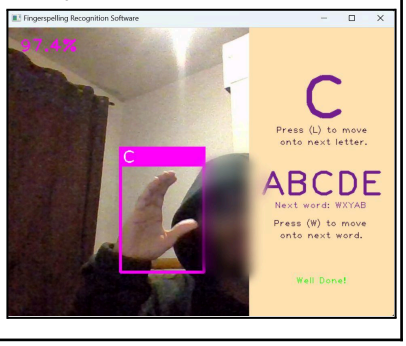
```
# Words
...
instructionSetConstant = 0
```

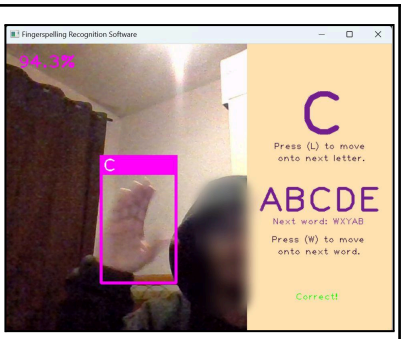
Additionally, if the hand is detected but positioned too far to the right (beyond the 320-pixel threshold), the application advises the user to move their hand to the left, ensuring optimal positioning for gesture recognition.

Following successful gesture recognition, the application randomly alternates between positive feedback messages ("Well Done!" or "Correct!"), using a simple randomisation mechanism to select the message displayed.

Testing Iteration 5:

Test		Result	
Type	**Input**	**Expected**	**Output**
Invalid	No hands detected before model is loaded	"Place hand in front of camera to begin." message displayed.	✓ "Place hand in front of camera to begin." message displayed. 

Normal	Hand position is out of predefined boundary	"Move hand on the left hand side of the screen." message displayed.	✓ "Move hand on the left hand side of the screen." message displayed. 
Boundary	Hand position is at the edge of the camera view and GUI.	"Move hand on the left hand side of the screen." message displayed.	✓ "Move hand on the left hand side of the screen." message displayed. 
Invalid	No hand detected	"Place hand in front of camera" message displayed.	✓ "Place hand in front of camera" message displayed. 
Normal	Correcte gesture for a letter	Random congratulatory message ("Well Done!" or "Correct!") displayed.	✓ Random congratulatory message ("Well Done!" or "Correct!") displayed 

Stakeholders Feedback

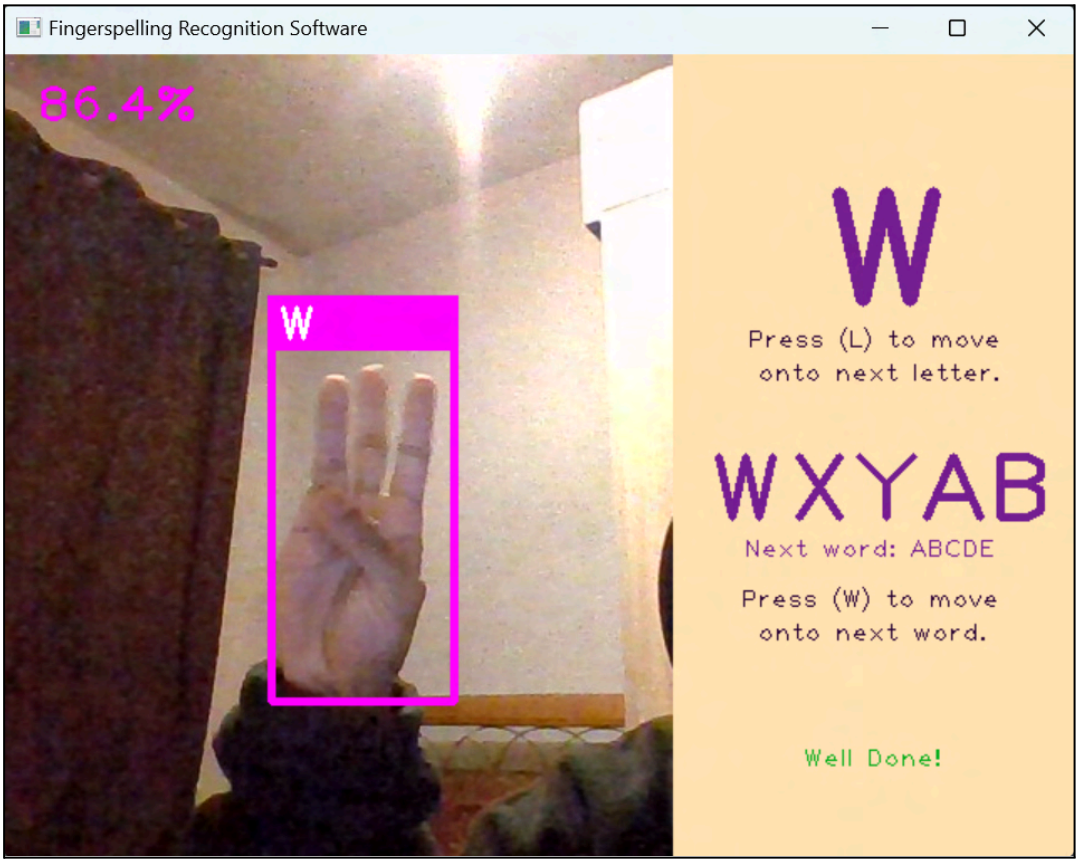
"I have made some changes to the GUI, so that everything fits together. What do you think of these changes?"

Yusuf *"Wow! I really like the way the program is so interactive now, through constant feedback."*

Robert *"Everything is perfect. Well Done!"*

Imtiaz *"That is what I wanted. However, could you please make the green message a bit more darker, they blend in with the background since they are both light colours."*

"Sure! I will do that."



"What do you think about this? I tried to make it a bit darker, not too dark so that it is easily distinguishable with the other text."

"Yes, that is fine."

Iteration 5 Review:
Displaying the proper instruction/message demonstrates thoughtful implementation of user interaction and feedback within a fingerspelling recognition application. They are structured to guide users through the process of hand placement and gesture recognition, enhancing the learning experience with real-time instructions and encouragement.

Below are all the specification points that I have met:

Requirement	Sub-Requirement	User Requirement	Success Criteria
Functionality	Instructions	Inform user to begin by placing hand in front of camera	- Display message *"Place hand in front of camera to begin."* when the user starts the program. - Remove message as soon as the hand is detected. - Message will be displayed underneath the word to be fingerspelled. - Message will be displayed in red.
		Inform user hand is not localised	- Display message *"Place hand in front of camera."* only if the model can't localise the hand. - Remove message as soon as the hand is detected. - Message will be displayed underneath the word to be fingerspelled. - Message will be displayed in red.
		Inform user to position hand on the left hand side of the screen	- Display message *"Move hand on the left hand side of the screen."* - Remove message, as soon as the whole bounding box is present on the left hand side of the screen. - Message will be displayed underneath the word to be fingerspelled. - Message will be displayed in red.
		Inform user correct hand spelling	- Display message *"Well Done!"* or *"Correct!"* if the user's sign matches with letter to be fingerspelled. - Message will be displayed underneath the word to be fingerspelled. - Message will be displayed in green.

Final Adjustments

Now I can decide which words my stakeholder will fingerspell, in `words.txt`. The choice of the following words are based on increasing difficulty, with initial words easy to fingerspell, and as the user progresses, the letter to fingerspell becomes more difficult to sign.

```
CABLE
MANGO
PANEL
GLOBE
MOULD
AMPLE
BOXED
NYMPH
VEXED
QUAKY
WAXEN
```

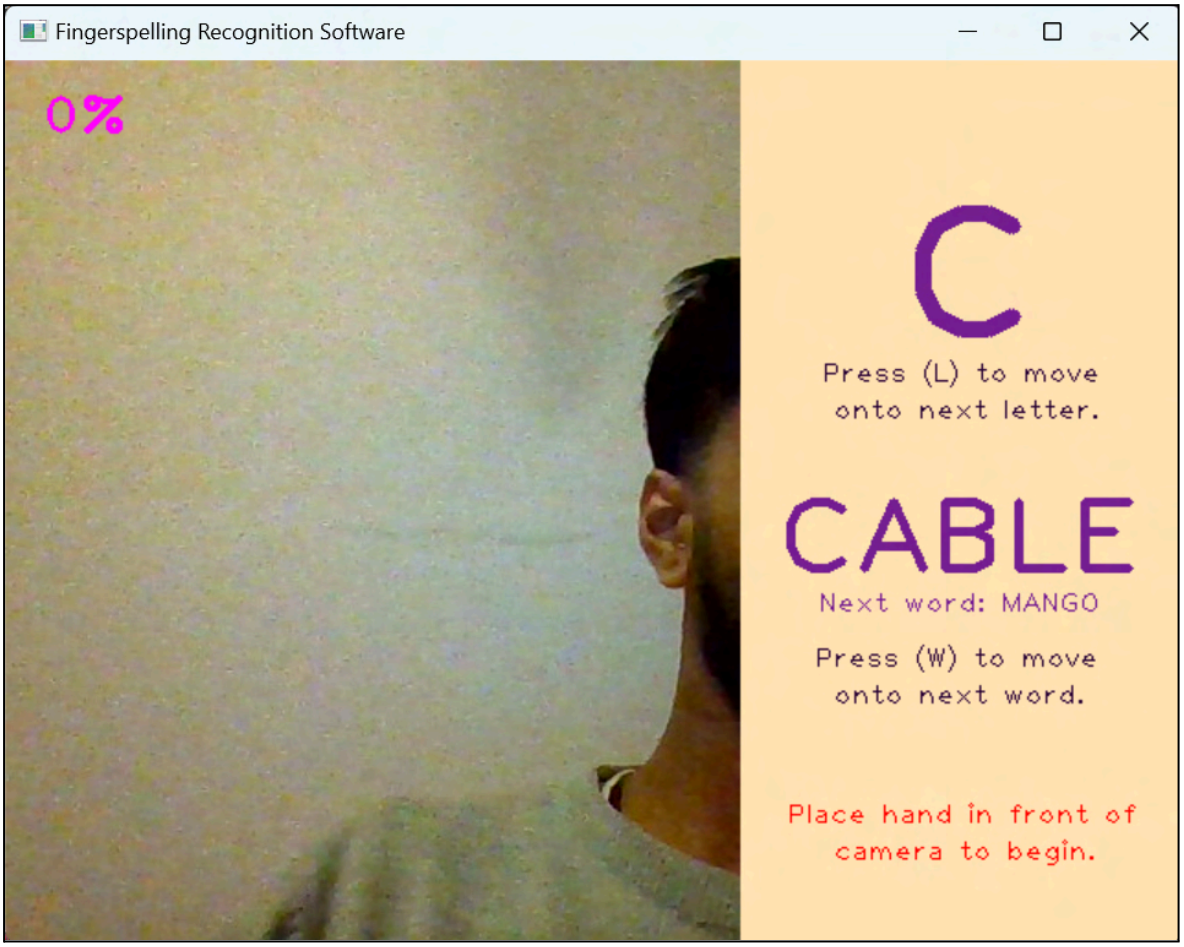
Post-Development Testing

The stakeholder and I will engage in black-box testing of the prototype's functionality, adhering to the Post-Development Testing table. I have invited stakeholders to test the software to capture their firsthand experiences with it. For each section in the table, I have analysed the final results and compared them to the expected outcomes listed.

Additionally, I have emailed my stakeholders specific questions about the software to collect feedback for the final evaluation of my product and to inform future enhancements.

Video Capture

Action	Expected Result	Related Success Criteria
Initiate software	The webcam is activated, and video feed begins	Users should experience no delay or issues in the initialisation of the video capture for hand recognition.
Actual Result		



When first initiating the Fingerspelling Recognition Software, the user is directly taken to the active practice window, eliminating any intermediary loading screens or menus. This design choice ensures that users can immediately begin their fingerspelling practise without unnecessary delays. The window displays a live video

feed from the user's webcam, allowing them to see themselves and ensuring they are correctly positioned for gesture recognition.

The fluidity and clarity of the video feed, along with the software's ability to recognise hand signs, are contingent upon the quality of the user's webcam. A high-quality webcam with greater frames per second (fps) and higher resolution will naturally provide a more accurate and responsive gesture recognition, contributing to a more effective practice session.

Stakeholders Feedback

"How do you find the initialisation process of the software when activating the webcam for sign language recognition? Are there any delays or issues you encounter that hinder the start of your fingerspelling session?"

Yusuf	*"I've been using the software to practise my fingerspelling, and I must say, the startup is pretty slick. No waiting around for the webcam to kick in, which is great because I like to dive straight into practising. However, there have been a couple of times when the feed didn't start on the first try, and I had to restart the application. It's not a big deal, but smoothing that out would make the experience even better. Other than that, the immediacy of the video feed helps me focus on my signing without any frustrating interruptions."*
Robert	*"The webcam activates swiftly each time I start the software, which is essential for me in my limited prep time before work meetings. I've noticed that in different lighting conditions the response time varies slightly. Under bright lights, it's instantaneous, while in dimmer settings, there seems to be a brief lag. If that could be optimised, it would ensure consistent performance regardless of where I'm practising. It's a minor thing but it makes a difference when you rely on the software for daily practice."*
Imtiaz	*"I'm pretty new to all this, but I found the setup process to be quite intuitive and hassle-free. The webcam turns on and the software is ready to go almost immediately, which is perfect for someone like me who's juggling learning sign language with a ton of other daily tasks. I did notice that my older webcam took a bit longer to get recognised by the software compared to my brother's newer one, so maybe there's something there that could be enhanced for users with older equipment. Other than that, it's been smooth sailing, and it really feels tailored to help me learn effectively."*

Stakeholders acknowledge the software's responsive initiation and appreciate the quick activation of the webcam, which aids in a smooth start to their fingerspelling practice sessions. They recognise the importance of this immediate feedback in maintaining focus and flow, particularly during continuous learning activities.

Yusuf notes the generally swift startup but mentions occasional instances requiring a restart. He suggests further smoothing the initiation process to eliminate even these rare interruptions. An improvement to the code could be implementing a more robust error-handling mechanism that attempts to reinitialise the webcam automatically if the first attempt fails. To solve this issue, I could introduce a retry loop with a delay and a maximum number of attempts for webcam initialisation. If initialisation fails on the first try, the software can wait for a short interval before attempting again.

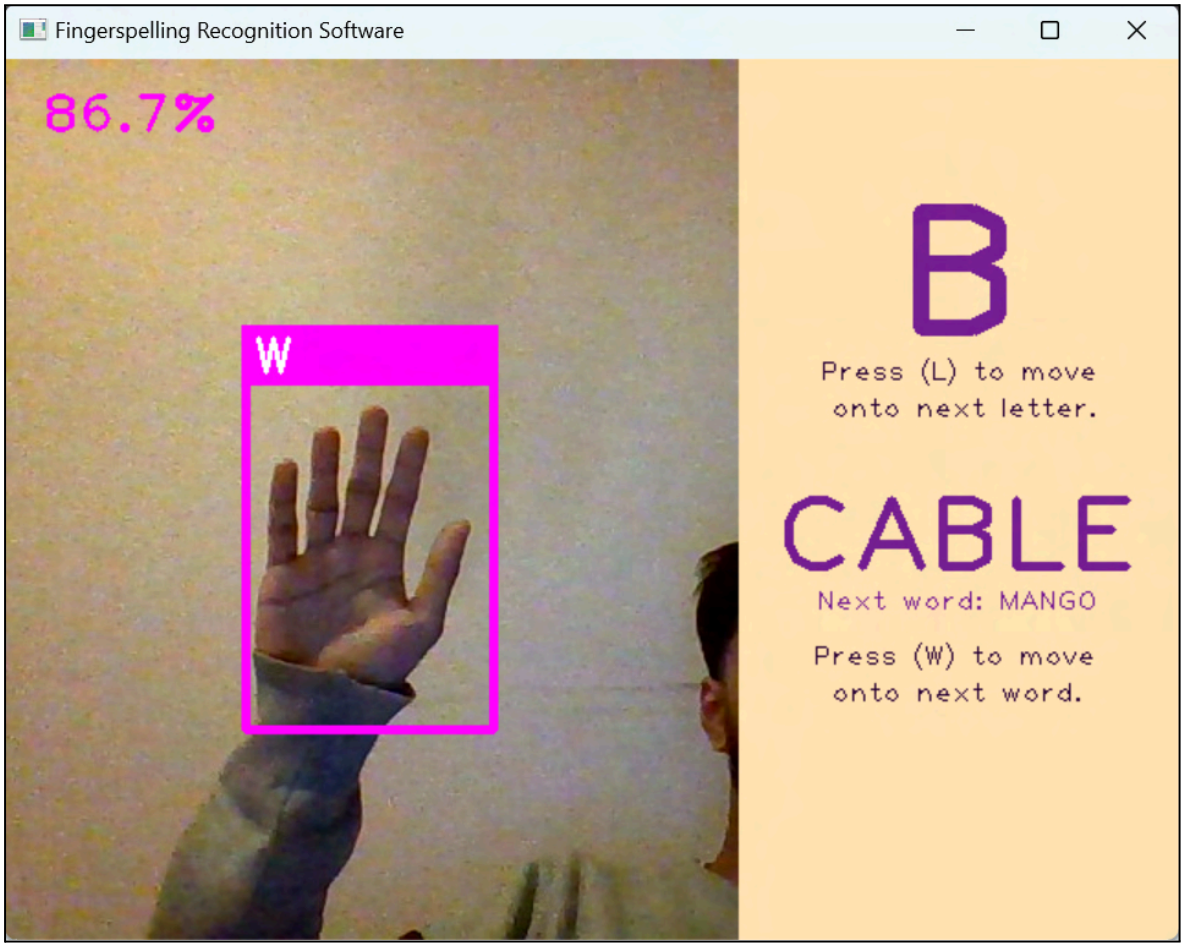
Robert appreciates the quick activation under bright lighting but points out a slight delay in dimmer conditions. To address this, the code could be optimised to adjust camera settings based on the lighting conditions, perhaps by auto-adjusting the exposure or by prompting the user to increase ambient lighting if below a certain threshold. I could implement automatic exposure control or guide the user to improve lighting if the environment is too dark. This could involve real-time image analysis to assess brightness levels and suggest adjustments or apply image processing techniques to enhance visibility.

Imtiaz finds the setup intuitive and mentions a difference in response times depending on the webcam's quality. Enhancing compatibility with a range of webcam models and ages could involve incorporating a feature that detects and optimises settings for lower-quality cameras to reduce recognition latency. I could create a calibration tool within the software that tests and adjusts settings based on the detected camera model's performance.

Hand Detection

Action	Expected Result	Related Success Criteria
One hand is present within the video feed	Hand is detected and a bounding box appears	Users should have their hand detected consistently for gesture recognition to take place.

Actual Result



Upon successful hand detection within its operational frame, the Fingerspelling Recognition Software dynamically overlays a bounding box around the user's hand, which actively tracks and adjusts to the hand's movements. This bounding box is a critical component of the interface, serving as a visual confirmation that the hand has been properly identified and that the software is ready to analyse the specific gesture being made. It is crucial for the user to maintain clear hand movements within the designated frame; otherwise, the software may struggle to accurately localise the hand, and consequently, the bounding box may not appear. This could impede the gesture recognition process.

To ensure precision in sign interpretation, the system is configured to recognise only one hand at a time. This focused approach prevents confusion from multiple inputs and contributes to the reliability of the gesture identification. The software's ability to discern and evaluate hand gestures is foundational to providing users with a seamless and interactive learning experience, allowing them to engage confidently with the fingerspelling practice.

Stakeholders Feedback	
"Can you describe your experience with the hand detection feature? How consistently does the software recognise your hand and display the bounding box to facilitate gesture recognition?"	
Yusuf	*“Hand detection works like a charm most of the time! I put my hand up, and there's the bounding box. But I've noticed it sometimes struggles with different background colours. Like, if I’m practising in my room with posters on the wall, it takes a second or two longer to figure out where my hand is. Consistency is key for me to learn, so maybe the software could be tweaked to handle complex backgrounds better? That would really make my practice sessions more about learning the signs and less about finding the right spot to place my hand.”*
Robert	*“I’m quite impressed with how the software picks up my hand movements. At the office, even with my desk cluttered, the bounding box latches onto my hand immediately, which is great. There have been rare occasions when I've had to move my hand a bit more into the centre to get it detected, but once it's there, the tracking is solid. For professional settings like mine, this reliability is crucial. If it can maintain this performance even as I shift positions, that would make it outstanding.”*
Imtiaz	*“The hand detection is generally spot on! Put your hand up, and the software does the rest, which is fantastic. Sometimes though, if the lighting isn’t great, like in the evenings, it doesn't pick up my hand straight away. If the software could handle low-light conditions a bit better, it'd be a huge plus. It’s not a deal-breaker by any means, but it's something I noticed. The bounding box is a cool visual cue, and once it's there, following along with the gestures feels really intuitive.”*

Stakeholders provide positive feedback on the hand detection feature's performance, with particular attention to its ability to facilitate gesture recognition.

Yusuf experiences occasional delays when background colours and patterns are complex, suggesting a need for the software to better differentiate between the hand and variable backgrounds. To enhance this, the code could incorporate adaptive background subtraction techniques or machine learning models trained on a more diverse dataset to improve hand recognition in complex environments.

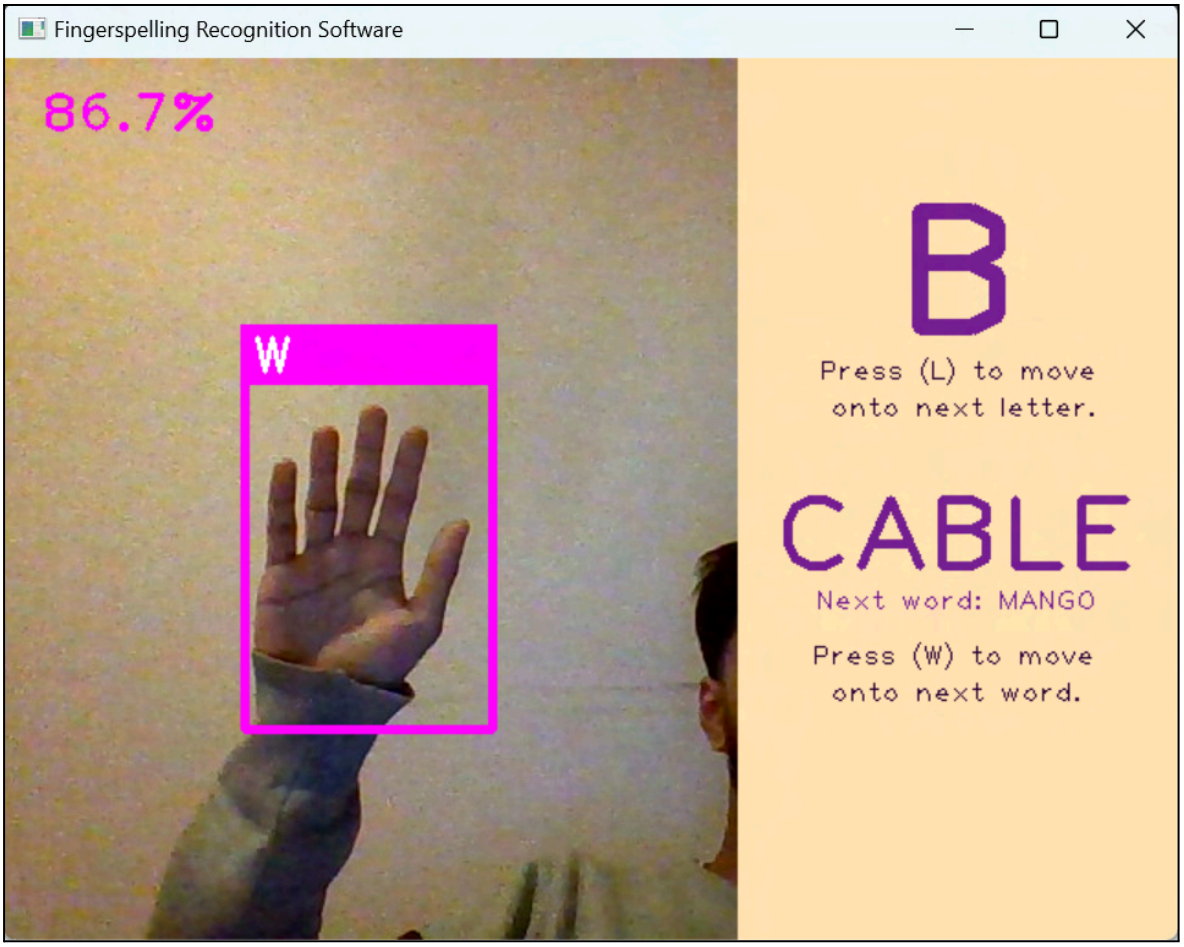
Robert is impressed with the hand detection in cluttered spaces but notes rare instances requiring hand repositioning for detection. Incorporating a feature that provides visual guidance on optimal hand placement can help users position their hands correctly, improving detection without the need to move around.

Imtiaz highlights the effectiveness of hand detection but mentions that low-light conditions can affect performance. The code could be improved by integrating a feature that evaluates the current lighting and automatically adjusts the webcam settings or provides user prompts to adjust their environmental lighting, ensuring consistent hand detection performance regardless of lighting conditions.

Classification of Hand Gesture

Action	Expected Result	Related Success Criteria
Hand performs a sign from the ASL alphabet.	Correct letter from ASL is recognised	Users should receive immediate and accurate recognition of their hand gestures corresponding to the ASL alphabet.

Actual Result



The Fingerspelling Recognition Software displayed in the image provides instantaneous feedback, actively attempting to interpret the hand gesture presented by the user. It assesses the hand's position and finger alignment present inside the bounding box, against the American Sign Language (ASL) alphabet database.

If the user's hand position doesn't precisely match a known ASL sign, the system employs its best approximation to determine which ASL letter the gesture most closely aligns with. In the example shown, the user's hand is raised, with the index, middle, and ring fingers extended upward, a recognised gesture for the letter "W" in ASL. The software accurately identifies this sign and displays an 86.7% confidence level for the gesture's match to the letter "W," visually confirmed by the highlighted bounding box around the hand.

This real-time analysis ensures that users receive immediate validation of their signing, enhancing the learning experience by reinforcing correct gestures and providing cues for adjustment where necessary.

Stakeholders Feedback	
"How effective do you find the gesture recognition feature in identifying the correct letters from the ASL alphabet during your use? Is the recognition immediate and accurate in your experience?"	
Yusuf	*“The gesture recognition is seriously impressive. It almost always gets my signs right on the first try, which is super encouraging for my practice. There's hardly any delay, so I can move through signs pretty quickly. That being said, some letters, especially those that require more intricate hand positions, can be a tad hit-or-miss. It would be awesome if the software could be even more fine-tuned to pick up on those subtle differences because that immediate feedback is what really helps me polish my skills.”*
Robert	*“I find the recognition technology to be quite proficient, particularly with straightforward signs. The immediacy with which it recognises most letters allows for a fluid practice session, which is invaluable when I'm preparing for communication in meetings. The accuracy does falter occasionally with more complex signs, where precision is critical. Enhancing the software's sensitivity to these nuances would be beneficial. Overall, though, it's a robust feature that has become a staple in my daily routine.”*
Imtiaz	*“The recognition of signs is pretty slick! I make a sign, and boom, the software displays the letter. It's gratifying to see that instant recognition, and it helps reinforce my learning. Occasionally, I’ve noticed that if I don't position my hand just right, it might confuse one letter for another. If the recognition could be made even more forgiving to slight variations in hand positioning, it'd be perfect for beginners like me who are still getting the hang of precise signing.”*

Stakeholders Yusuf, Robert, and Imtiaz provide valuable insights into the gesture recognition feature of the software, emphasising its overall effectiveness.

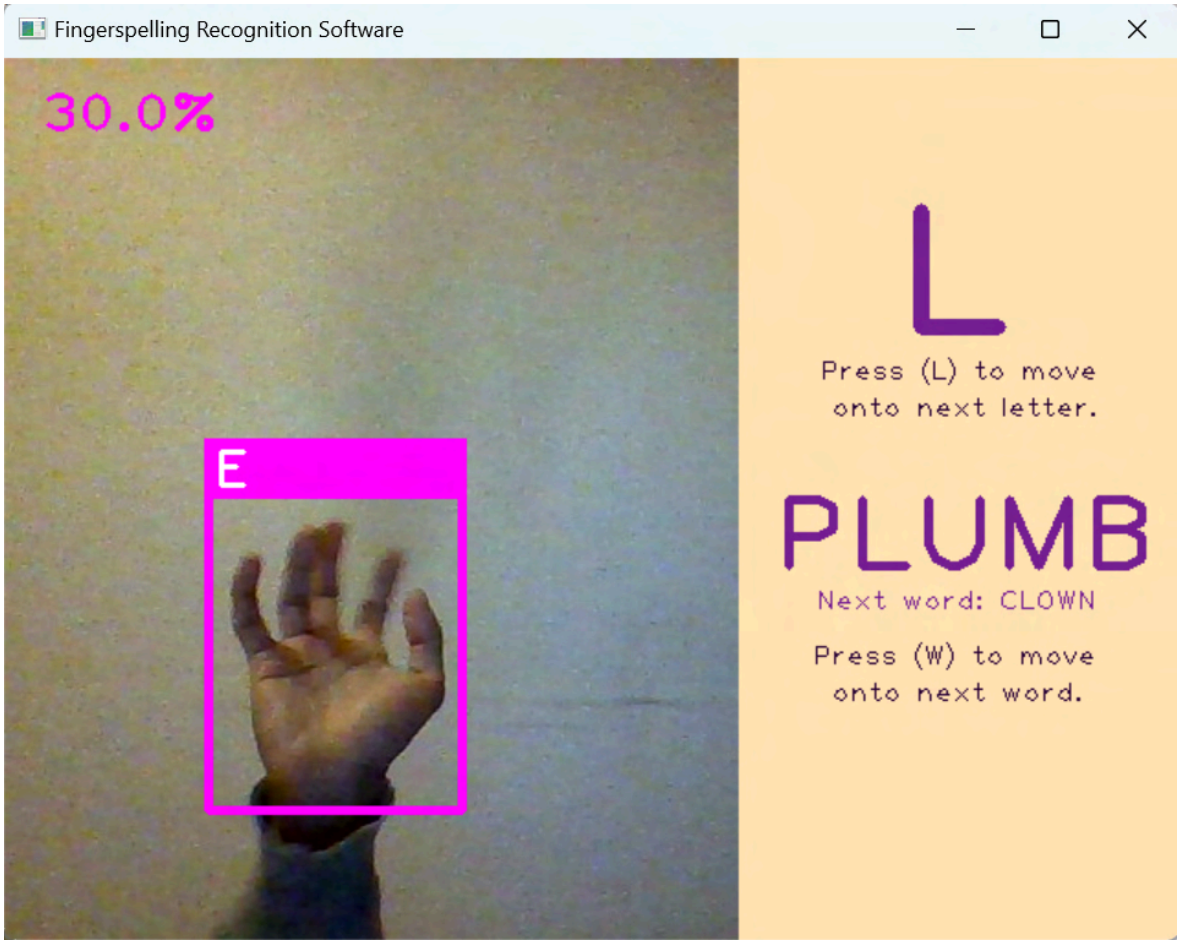
Yusuf is impressed with the system’s quick recognition of gestures but mentions that certain ASL letters that require more nuanced hand positions can be challenging for the system to recognise. This feedback suggests that the code could benefit from the integration of a more sophisticated gesture recognition algorithm, perhaps utilising a deeper neural network capable of capturing the finer details of hand positions.

Robert finds the gesture recognition to be proficient and appreciates the immediacy with which it responds to most letters, crucial for his professional use. Yet, he points out the occasional difficulty with complex signs where precision is essential. Enhancing the software to better handle these nuances could involve training the recognition model on a broader range of hand shapes and positions to increase its accuracy.

Imtiaz enjoys the positive reinforcement from correct sign recognition but has noticed that if his hand is not positioned perfectly, the software might misidentify the sign. To make the software more beginner-friendly, as Imtiaz suggests, the recognition process could be adjusted to allow a slight margin of error in hand positioning. This could be done by relaxing the model's parameters or by applying post-processing to the model's output to account for common errors in hand placement by beginners.

Model Prediction Accuracy

Action	Expected Result	Related Success Criteria
Model recognises the sign	Current prediction accuracy is displayed as a percentage	Users should be provided with real-time feedback on the accuracy of their gestures to facilitate self-assessment.
Actual Result		

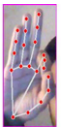
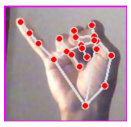
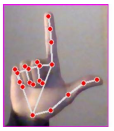
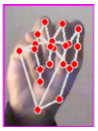
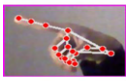
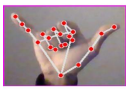


In the image, the software is showcasing its real-time analysis capabilities, displaying a confidence rate in the upper left corner that reflects how closely the user's hand gesture matches a specific letter from the ASL alphabet. The software has identified the user's gesture as the letter "E" with a confidence rate of 30.0%. However, it's clear that the intended gesture is for the letter "L," as indicated by the large displayed icon and the current word "PLUMB" to the right, suggesting a discrepancy in the recognition.

The displayed confidence rate is crucial feedback for the user, as it quantifies the model's certainty in its prediction, thereby providing an opportunity for the user to adjust their hand positioning to improve accuracy. This feature is particularly beneficial for learning and perfecting the art of fingerspelling, as it allows the user to receive instant feedback and make real-time corrections to their signs.

The performance of the fingerspelling recognition system is measured by the confidence rates assigned to each detected ASL alphabet sign. Based on the training results, the system has achieved an average confidence rate of 95.33%. This metric reflects the system's ability to accurately recognise and interpret each hand gesture as a

specific letter from the ASL alphabet. Detailed in the table below, the confidence rates for individual letters are systematically recorded, providing a view of the system's efficiency in classifying each sign.

A	B	C	D	E	F	G	H
							
98%	94%	95%	99%	89%	98%	97%	87%
I	**K**	**L**	**M**	**N**	**O**	**P**	**Q**
							
85%	91%	96%	89%	90%	98%	99%	93%
R	**S**	**T**	**U**	**V**	**W**	**X**	**Y**
							
92%	86%	88%	94%	96%	98%	92%	99%

As shown, some of the confidence rates are relatively low compared to other ones. This may be because of multiple factors, for example the letter “S” and “E” closely resembles the sign for the letter “A”, and therefore the recognition model might mistakenly predict the wrong sign. The system's recognition capability and, consequently, its confidence rates could be further improved by enlarging the training dataset. A more extensive dataset would provide a broader range of examples for the model to learn from, enhancing its ability to correctly identify and interpret a wider variety of hand gestures, thereby refining the overall performance of the system.

Stakeholders Feedback	
"Can you share your thoughts on the accuracy percentage feature that displays after the software recognises a sign? Does this feature effectively aid you in real-time self-assessment during your practise?"	
Yusuf	*“The accuracy percentage is super helpful for tracking how well I'm doing. It's motivating to see the numbers go up as I get better at each sign. Sometimes, the percentage fluctuates, which confuses me a bit—does it mean my sign was close but not quite there? It'd be helpful if there were some tips or a visual guide that pops up when my accuracy isn't 100%, so I can understand what to adjust. But overall, it's a great feature that really helps me push to improve.”*
Robert	*“I find the instant accuracy feedback quite valuable—it provides a clear indication of how precisely I'm signing. It's a clever way to foster self-improvement. However, the percentage seems to be based on very fine margins, and I'm wondering how it accounts for the inherent variability in human hand movements. Perhaps a threshold setting would be beneficial, allowing users like me to set a personal accuracy goal. This customisation could adapt the learning process to individual needs, enhancing the software's utility in professional or casual settings.”*
Imtiaz	*“Seeing the accuracy percentage right after I make a sign gives me immediate insight into how I'm doing, which is awesome. It's like getting instant feedback without having a teacher there. What would make it even better, though, is if the software could give some hints or corrections when my accuracy isn't too high. That way, not only do I know that I need to improve, but I also get guidance on how to do it. It's been a valuable feature for my learning process for sure.”*

Stakeholders Yusuf, Robert, and Imtiaz find the accuracy percentage feature to be a significant aid in their learning process, providing immediate and motivational feedback.

Yusuf appreciates the feature for tracking progress but would benefit from additional guidance when his accuracy is not 100%. He suggests visual cues or tips for improvement could be incorporated. For this I could integrate an assistant feature that provides visual tips or corrections when the user's accuracy drops below a certain level. For example, if the accuracy is less than 100%, display a visual cue indicating what part of the sign needs adjustment.

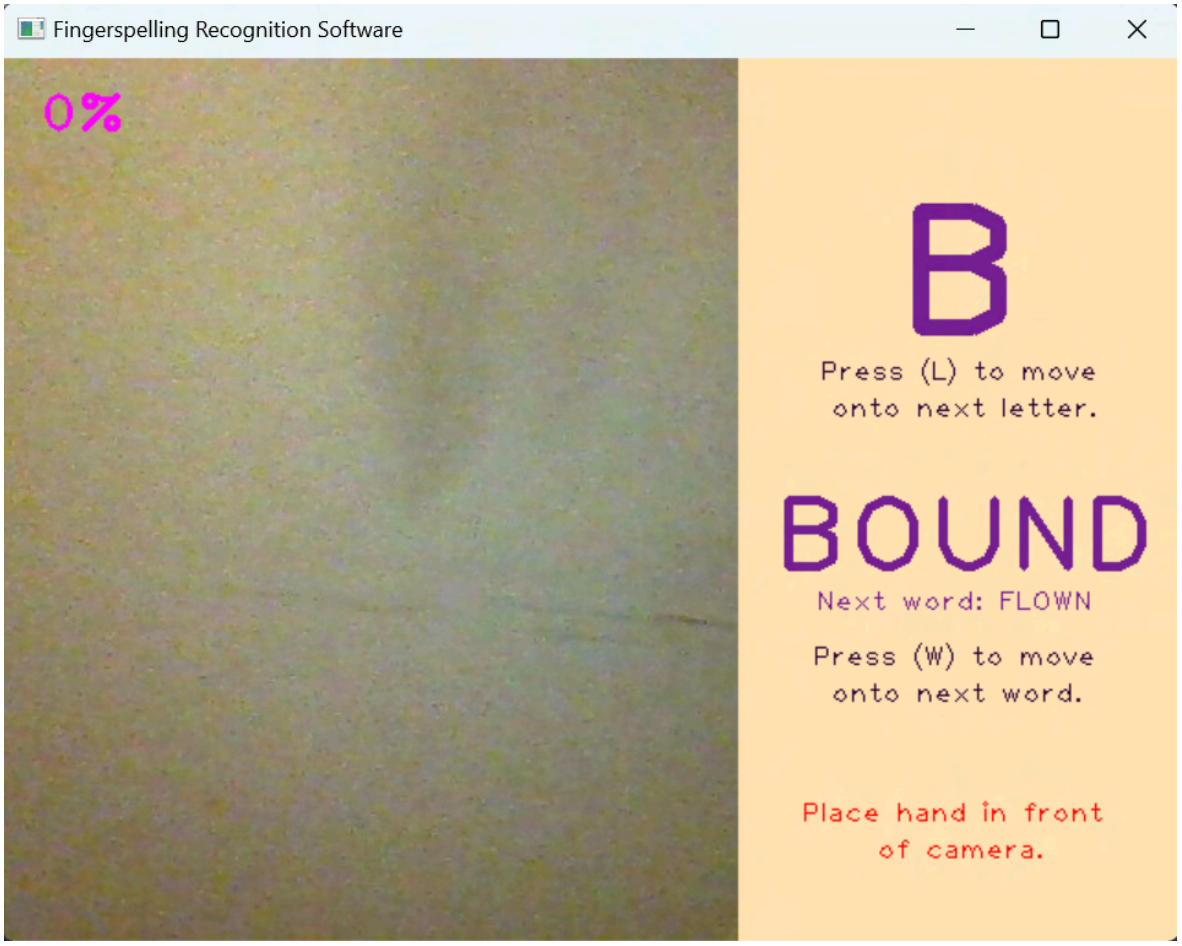
Robert values the instant feedback on precision but is curious about how the software accounts for natural variability in hand movements and suggests a customisable accuracy threshold. I could introduce a setting that allows users to set their own accuracy goals. The software could offer a slider or input field where users define what accuracy percentage they aim for, allowing for personalised difficulty levels.

Imtiaz sees the immediate accuracy feedback as essential, indicating a desire for instructive feedback when accuracy falls short. So to solve this, perhaps I should develop a feature that provides suggestions or interactive tutorials when the user makes an incorrect sign, helping them understand how to correct it in real time.

Instructional Messages

Action	Expected Result	Related Success Criteria
User does not know how to use software	Appropriate instructional messages are displayed depending on user interaction	Users should be guided by clear, contextual instructions to enhance understanding and usability.

Actual Result



The software interface implements instructional messages to guide users through their interaction with the system. These messages, designed to appear at specific moments, serve as navigational aids and troubleshooting prompts, enhancing the user experience by providing clear directions for corrective actions.

The message "Place hand in front of the camera to begin." appears as an initial prompt, indicating that the software is ready but awaiting the user's action to start the recognition process. This message typically displays when the software has been launched, and the model responsible for hand recognition is fully loaded and ready for input.

If the user has initiated the session but their hand is not within the camera's field of vision, a friendly reminder, "Place hand in front of camera." appears. This prompt reinforces the need for the user's hand to be visible to the camera for the software to begin the recognition process.

Together, these messages play a vital role in making the software more user-friendly, reducing frustration and encouraging users to make adjustments that will lead to a more successful experience with the software.

Stakeholders Feedback	
"How helpful do you find the instructional messages within the software when you're unsure how to proceed? Do these messages enhance your understanding of how to use the software and improve usability?"	
Yusuf	*“The instructional messages pop up just when I need them, and they're really clear. It’s great that they're tied to what I'm actually doing, so I don't feel lost at any point. Once, I wasn't sure how to go to the next word, and there it was, the instruction I needed. For someone who is eager to learn quickly, that's a big plus. Making those instructions even more interactive could be fun, like having them highlight areas I need to click or gesture towards.”*
Robert	*“I appreciate that the software anticipates where I might need some hints and provides instructions accordingly. It makes the experience feel very user-friendly and accessible. The messages are concise, which is essential for me since I can’t afford to spend too much time figuring things out during my workday. The visual design of the messages is quite unobtrusive as well, blending well with the rest of the interface.”*
Imtiaz	*“The way the software guides you through with instructions is pretty neat—it’s like a built-in tutorial that’s always on standby. As a beginner, this has made the learning curve much less steep.”*

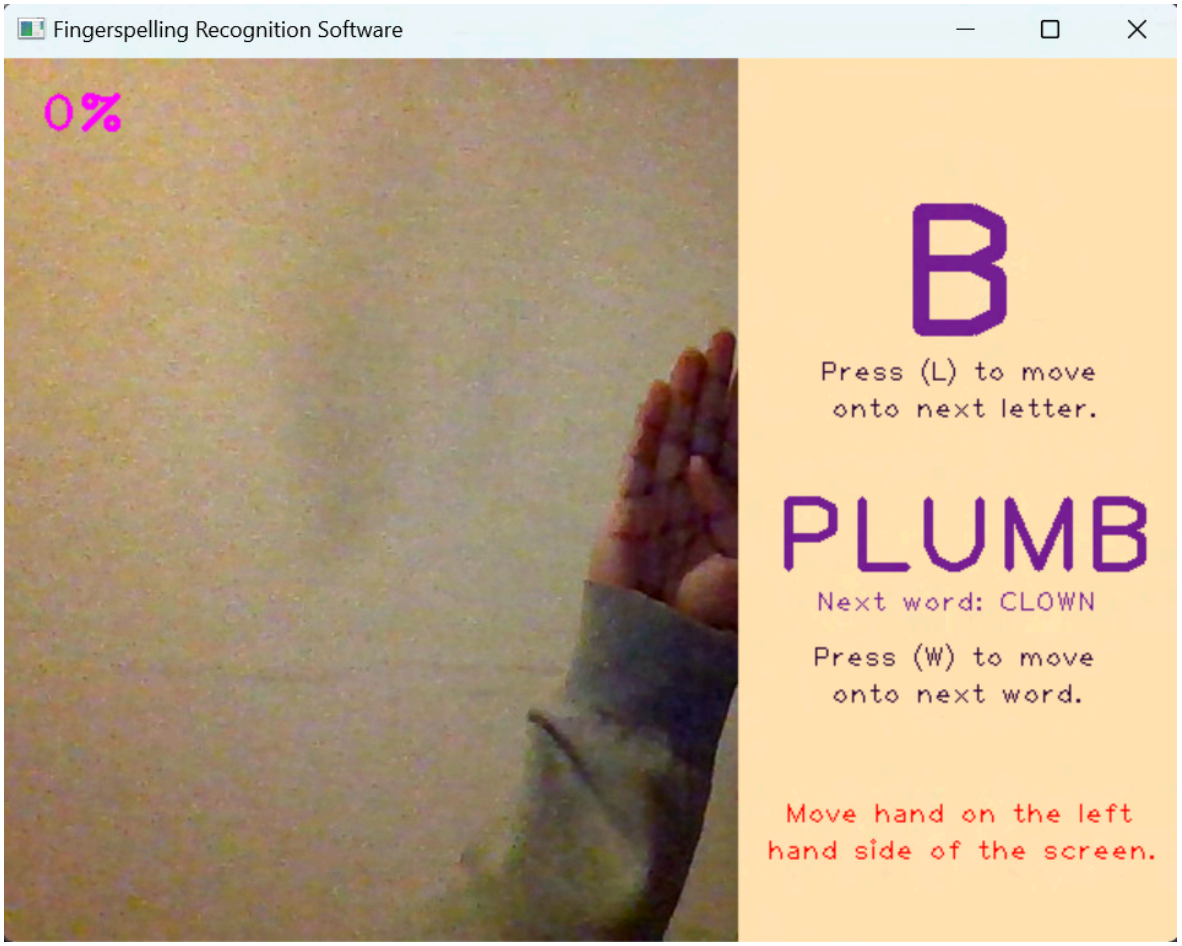
The stakeholders Yusuf, Robert, and Imtiaz provide positive feedback on the instructional messages within the software, praising their timeliness and clarity.

Yusuf finds the messages very helpful, especially when transitioning to new content, and suggests enhancing the interactivity of these messages, such as highlighting actionable items on the interface. For Yusuf's suggestion, the software could implement interactive elements such as animated highlights or arrows that draw attention to interface areas requiring user action.

Error Handling for Hand Detection

Action	Expected Result	Related Success Criteria
Hand is detected, but it's not in the predefined frame.	When hand is outside the defined frame, the system prompts the user to move their hand into view	Users should be alerted when their hand is not correctly positioned, ensuring they are always ready for gesture recognition.

Actual Result



In the interactive environment of the software, as depicted in the attached image, the system is adept at detecting the user's hand against the varying backdrop of the live camera feed. When the hand is recognised but not clearly visible due to overlap with the GUI elements, a clear on-screen message, "Move hand on the left-hand side of the screen," instructs the user to reposition their hand.

This guidance is crucial as the bounding box that highlights the detected hand is layered beneath the GUI, which might make it difficult for users to know whether their gestures are being accurately captured and interpreted by the system. By prompting users to move their hand to the left, the software ensures the hand remains in clear view, not only for the user's benefit but also to maintain optimal functionality of the hand detection algorithm. This kind of smart feedback is essential for maintaining a smooth user experience and helping users adjust their actions to fulfil the software's operational requirements.

Stakeholders Feedback	
"Have you found the system's prompts effective when your hand moves outside the designated area for recognition? Do these alerts help you in repositioning your hand to ensure it's ready for accurate gesture recognition?"	
Yusuf	*"Yeah, the prompts are really on point. If my hand drifts off to the side or too close to the camera, the software tells me right away, and I can fix it. It's good because I don't want to learn the signs wrong by practising them out of frame. But if I'm being honest, the alerts could be a bit more noticeable. Sometimes I'm so focused on my hand that I don't notice the prompt immediately. Maybe if it flashed or was a different colour, it would catch my eye faster."*
Robert	*"The prompts for repositioning are subtle yet effective. They maintain the flow of my practice by ensuring I'm in the right spot for the software to recognise my signs accurately. In a busy office, these gentle nudges keep me from making errors due to misplacement of my hand. The prompt's timing is good; however, I believe the visual cue could be more prominent, perhaps with an animation or an outline of the correct positioning on screen to guide my hand back into place."*
Imtiaz	*"I get a little reminder from the software whenever my hand isn't where it should be, and that's super helpful. It keeps me from practising the signs wrong, which is important when you're trying to learn something new. But sometimes I wish the prompt was a bit clearer, like maybe it could show me an outline of where my hand needs to be? That'd make it easier to adjust without guessing if I'm back in the right spot."*

The stakeholders Yusuf, Robert, and Imtiaz have shared their experiences with the system's prompts for hand positioning.

Yusuf praises the prompt's accuracy but suggests enhancing visibility, such as flashing or changing colours, for immediate notice. For Yusuf's feedback, introducing more attention-grabbing visual elements in the code would be beneficial.

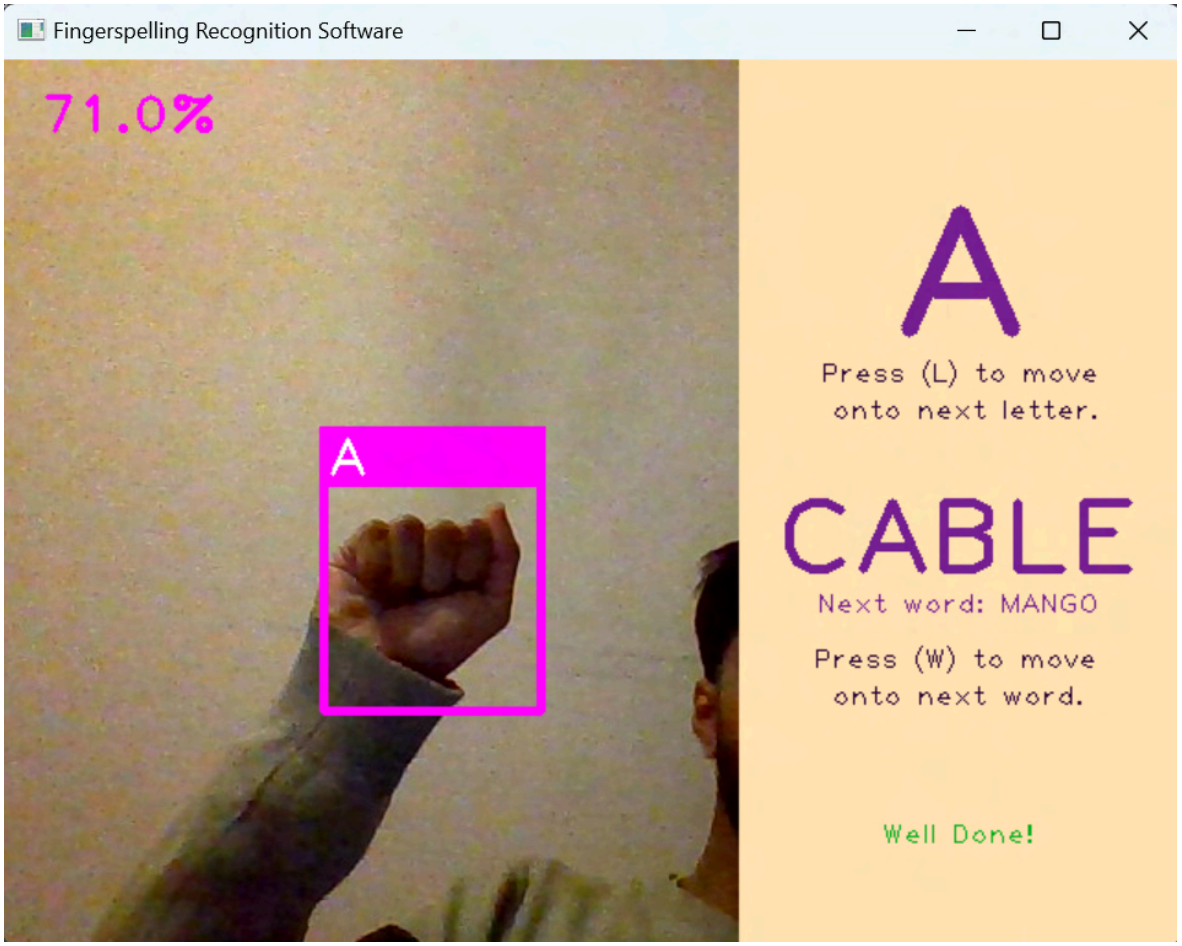
Robert finds the prompts helpful and appropriately timed but proposes more prominent visual cues, like animations, for guiding the hand back into place. To address Robert's suggestion, implementing an animation that guides the user's hand back to the correct position could enhance the software.

Imtiaz appreciates the reminders which prevent practising incorrectly, and he expresses a desire for an on-screen outline showing where the hand should be positioned. And for Imtiaz, incorporating a visual outline of the hand placement area would provide direct, actionable feedback.

Recognised Letter matches the Target Letter

Action	Expected Result	Related Success Criteria
Hand sign recognised by the model matches the target letter	The system advances to the next letter or word	Users should be able to proceed to the next letter or word upon successful recognition, enabling progressive learning.

Actual Result



In the software, as seen in the attached image, when the user successfully signs a letter that the model recognises with sufficient confidence, positive reinforcement is immediately provided. This is a key feature designed to encourage the learning process. For instance, the software has identified the gesture as the letter "A" with a 71.0% confidence rate. This real-time feedback allows the user to adjust and improve their signs dynamically.

The software then automatically progresses to the next letter within the word "CABLE" to continue the learning sequence. The message "Well Done!" appears on the screen, serving as an affirmation of the user's correct gesture, fostering a sense of achievement and motivation to continue practising.

The image suggests that absolute precision (100% accuracy) is not necessary for the software to recognise the sign and allow the user to proceed. This design choice reflects a balance between precision and usability, accommodating natural variability in users' signing.

Introducing a customisable accuracy threshold could be a beneficial enhancement, offering users the flexibility to set their own difficulty level or accuracy requirements. Such a feature would personalise the learning experience, making the software adaptable to users with different skill levels and learning objectives.

Stakeholders Feedback	
"When you correctly perform a sign and it's recognised by the model, how seamless is the transition to the next letter or word for you? Does this feature support your progressive learning by allowing you to move forward without disruption?"	
Yusuf	*"It's pretty cool how the system just flows to the next sign after I get one right. It keeps the momentum going, which is crucial for staying in the zone while practising. There was this one time when it hesitated a bit before moving on, but that's rare. Mostly, it's like a smooth conversation from one sign to the next. Maybe adding a little animation or sound when transitioning would make it even more engaging, kind of like a little celebration for getting it right."*
Robert	*"The transition mechanism is quite fluid, which is important for maintaining the rhythm of my practice sessions. The automatic advancement to the next letter after correct recognition is exactly the kind of functionality that reinforces my learning. It's gratifying and motivates me to continue. One suggestion might be to incorporate an option to pause on a sign for review or practise it repeatedly before moving on, giving users control over their learning pace."*
Imtiaz	*"I really like that it doesn't just stop after recognising a sign; it takes me to the next one without me having to do anything, which is super convenient. This way, I can keep going through the signs and really get into a learning flow."*

Stakeholders Yusuf, Robert, and Imtiaz provide feedback that endorses the software's transition feature between signs, which facilitates uninterrupted progressive learning.

Yusuf admires the smooth flow but suggests the experience could be enhanced with animations or sounds during transitions for a celebratory feel. Addressing Yusuf's feedback could involve adding a celebratory animation or sound effect within the code when a sign is correctly recognised.

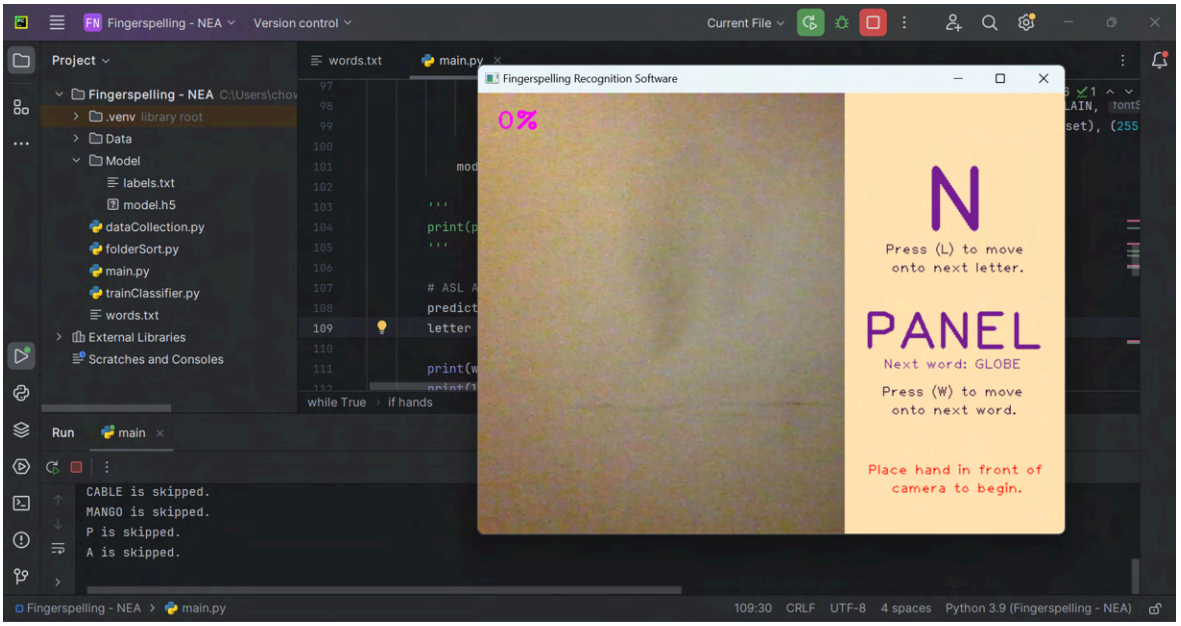
Robert is pleased with the fluid transition and suggests adding a pause function for additional practice on a sign, which would offer more control over the learning pace. For Robert's suggestion of a pause function, the code could be modified to include a toggle state that either advances automatically or waits for user input, giving the user more control

Imtiaz values the seamless advancement to the next sign, emphasising the convenience it brings to his learning process. To facilitate Imtiaz's seamless learning experience, ensuring that the current feature that automatically moves to the next sign is maintained and optimised is crucial.

Additionally, feedback from all stakeholders could lead to the creation of a settings menu where users can customise their transition experience, such as turning on or off automatic advancement or selecting celebratory feedback options.

Skip Functionality

Action	Expected Result	Related Success Criteria
Appropriate skip button is pressed	The software skips the current letter or word and advances as intended, if respective button is pressed	Users should be able to control their learning pace by skipping letters or words as needed.
Actual Result		



The software's interface is designed to allow users with direct control over their learning sequence. As illustrated in the image, pressing the (L) key allows the user to skip to the next letter in the sequence, while the (W) key advances the user to the next word. The current word, “PANEL,” and the subsequent word, “GLOBE,” are clearly visible, providing an immediate visual cue for the user's progress.

In the development environment shown alongside the GUI, the console output registers each skip action, confirming that the software is responding to user inputs as intended. When the user presses either key, the software reacts instantaneously, updating the display without delay. This interaction is crucial, ensuring that the user can fluidly move through the practice session, focusing on specific letters or words as needed.

This skip feature is particularly useful as it accommodates various learning styles and paces. Users who may want to spend more time on difficult letters have the option to do so, while others can quickly move through familiar material. The system's feedback loop, visible in the console output, reinforces the user’s actions, enhancing the interactive experience and promoting a user-centric approach to learning.

Stakeholders Feedback	
"How effective do you find the skip button functionality in controlling your learning pace? Does pressing the skip button reliably advance you to the next letter or word as intended?"	
Yusuf	*"The skip button is a lifesaver when I'm practising and hit a letter that I'm already comfortable with or just not in the mood to work on. It responds instantly, and I can move on to the next one without any hitches. It's empowering to have that control over my session. Maybe the button could be a bit more prominently placed on the interface, though, so I can access it even quicker without looking for it."*
Robert	*"I use the skip button quite frequently, especially during my quick review sessions in the mornings. It works seamlessly, letting me bypass letters I feel confident about, which is great for my productivity. The functionality fits perfectly with my needs, allowing me to focus on areas that require more attention."*
Imtiaz	*"The skip button functionality is straightforward and does exactly what it needs to—lets me skip over parts I don't need to focus on. It's really helpful because sometimes there are just some signs I want to practise more than others. It never misses, and the transition is smooth, which keeps my learning session flowing nicely. Perhaps you could add a function to go back to the previous letter, so I can focus on mastering in signing the letter more effectively."*

The stakeholders Yusuf, Robert, and Imtiaz provide positive feedback on the skip button feature, finding it crucial for managing their learning pace and navigating through the signs.

Yusuf values the control it gives him, suggesting an enhancement for even quicker access by making the button more prominent. To enhance Yusuf's experience, the software's user interface could be updated to make the skip button larger or more visually distinct.

Robert frequently uses the feature for efficient learning, implying that its seamless integration aligns with his need for focus. For Robert, maintaining the current skip functionality while considering his feedback about the feature fitting his needs well is essential. Ensuring that any UI changes do not disrupt his established workflow will be important.

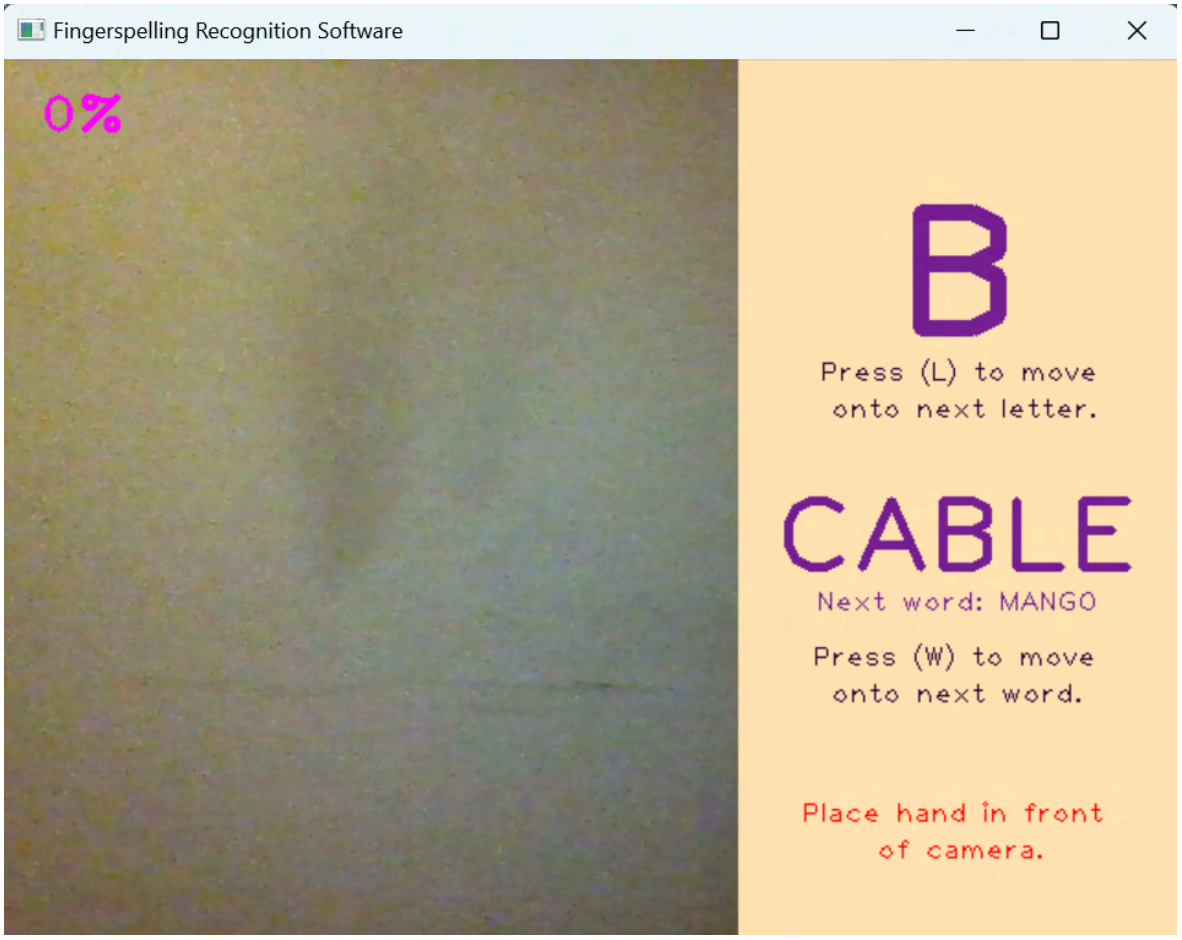
Imtiaz appreciates the straightforward functionality and smooth transitions, proposing the addition of a feature to revisit previous signs for further practice. To address Imtiaz's suggestion, the code could be adapted to include a back button or a gesture that allows the user to go to the previous letter or word:

```
# Example pseudo-code for revisiting previous signs
if key == ord("b"):  # Let's assume 'b' is the key to go back
    if currentLetterIndex > 0:
        currentLetterIndex -= 1  # Move back to the previous letter
    else:
        # Handle case where currentLetterIndex is at the beginning
```

Usability Testing

GUI Layout

Action	Expected Result	Related Success Criteria
User performs any action	Software is responsive to the user's action, by displaying corresponding results on the screen.	User is able to understand how to use the program, through continuous real-time feedback, allowing communication between user and software.
Actual Result		



The software's graphical user interface (GUI) presents a simple and user-friendly layout, facilitating an intuitive experience for users learning ASL fingerspelling. In the centre, the displayed letter “B” indicates the current letter the user is prompted to fingerspell. The software's recognition system is actively awaiting the user's gesture to match this target letter.

Directly below the target letter, the word “CABLE” is shown, providing context for the letter within a word, allowing users to practise and understand the flow of fingerspelling in actual communication. Below that there is the "Next word: MANGO", offering users a glimpse of upcoming practice words and the option to prepare ahead of time or skip forward if desired.

Instructional cues are thoughtfully positioned around the interface. The directive “Press (L) to move onto next letter” and “Press (W) to move onto next word” are clear and conveniently placed, guiding users through their practice with ease.

In the top left corner, a confidence percentage is displayed, signifying the software’s assessment of the user's fingerspelling accuracy. Here, it shows 0%, indicating that no hand has been detected in the frame. This immediate feedback mechanism is essential for users to correct their gestures in real-time, as it provides a quantifiable measure of how closely their sign matches the expected letter.

This GUI setup ensures that learners have all the necessary information within their field of vision to seamlessly interact with the software, progress through the learning modules, and refine their fingerspelling skills effectively.

Stakeholders Feedback	
"How responsive do you find the software when you perform any action? Does the immediate display of results on the GUI help you understand how to effectively use the program and enhance your interaction with the software?"	
Yusuf	*“Every time I do something, the software reacts right away, which makes it feel like it’s keeping up with me, not the other way around. The feedback is instant, which helps me learn how to use everything more quickly. It's like having a conversation where the software understands me. It would be even better if there was a feature where the software could give suggestions based on my frequent actions, sort of like a smart assistant.”*
Robert	*“The software’s responsiveness is impressive. Actions on my part, whether they’re gesture-based or through the interface, result in immediate visual feedback, which is crucial for maintaining an efficient workflow. This responsiveness not only aids in understanding the program’s functionality but also assures me that the system is working as expected.”*
Imtiaz	*“I'm quite pleased with how quickly the software responds to my actions. It’s very intuitive—feels like it’s designed to be user-friendly, especially for beginners like me. Every action I take results in an immediate response on the screen, which really reinforces my learning. Sometimes, though, I wish the feedback was a bit more detailed, especially when I make mistakes. It would be great if the software could not just react, but also guide me on how to correct errors in real time.”*

Stakeholders Yusuf, Robert, and Imtiaz commend the software's responsive nature, noting that it enhances their learning by providing immediate feedback to their actions.

Yusuf suggests that the software could further assist by offering suggestions based on user actions, like a smart assistant. This could be implemented by adding a feature that analyses the user's frequent mistakes and provides tips for improvement. Perhaps I could introduce an AI-based feature that tracks the user's progress and common errors. For instance, if a user frequently confuses signs with similar shapes, the software could highlight this pattern and provide targeted exercises to distinguish between them.

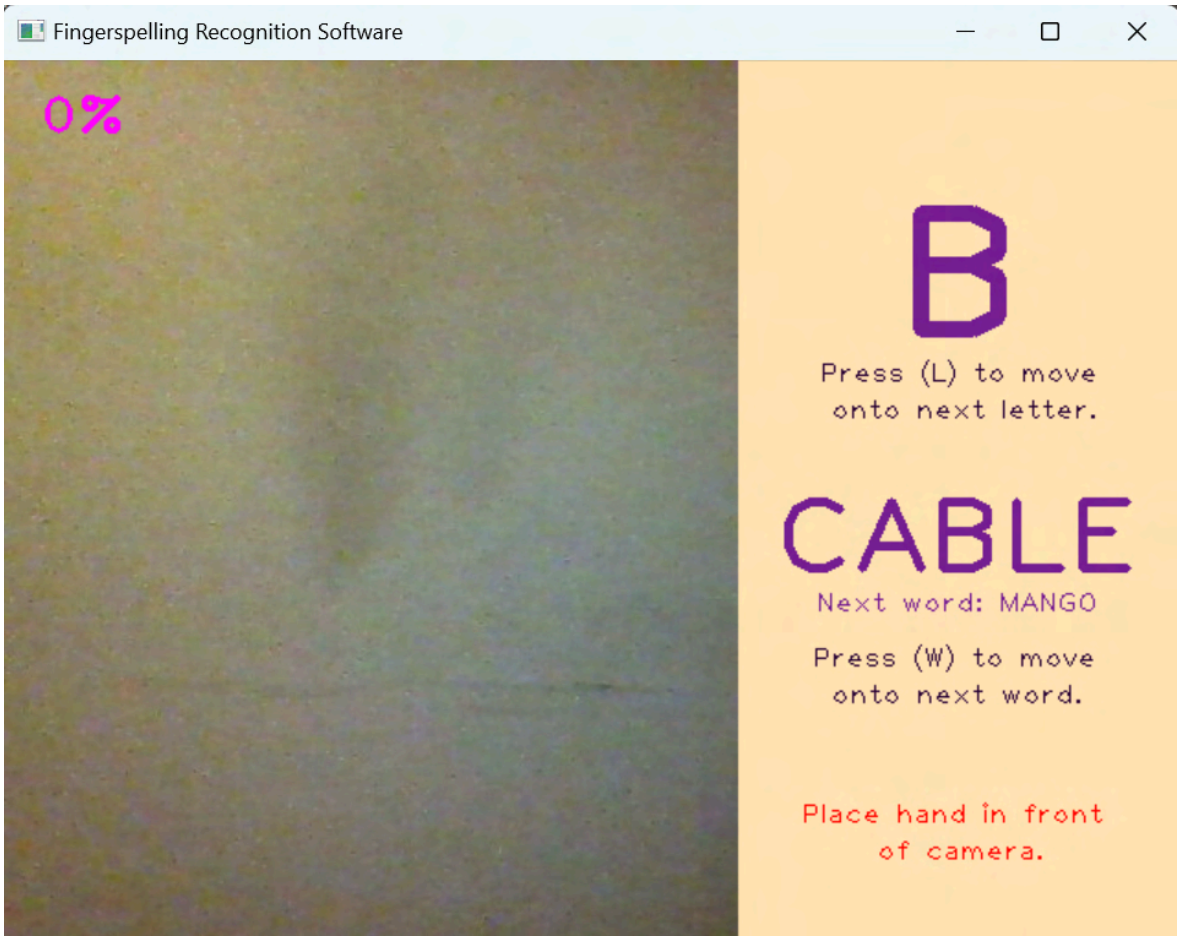
Robert is impressed with the software's responsiveness, which contributes to an efficient workflow and reassures him of the program's functionality. However, he does not specify any improvements, indicating satisfaction with the current state.

Imtiaz appreciates the user-friendly design and instant feedback but desires more detailed guidance when mistakes are made. To address this, the software could incorporate a real-time error correction guide that provides corrective suggestions when it detects a recurring error in the user's gestures. I can implement an interactive guide that activates during the learning session to offer real-time corrective feedback. This could be a sidebar or a tooltip that appears when the system detects an incorrect sign.

Navigation through Letters and Words

Action	Expected Result	Related Success Criteria
Navigation through Letters and Words	Users can cycle through the ASL alphabet and words for practise	Users should be able to navigate through letters and words seamlessly for an uninterrupted learning experience.

Actual Result



The software is designed to enable users to methodically cycle through the ASL alphabet and words, providing a structured learning path for both beginners and those looking to refine their skills. With the press of a designated key—(L) for advancing through the alphabet and (W) for moving to the next word—users can control their learning pace and sequence. This feature facilitates a smooth and customisable learning experience, as users can seamlessly transition between letters and words without the need to navigate complex menus or interfaces.

The cycling mechanism is crucial for reinforcing muscle memory and recognition of signs. It allows users to focus on letters or words that they find challenging and spend additional time practising without disruption. The ability to anticipate and prepare for the next letter or word helps users build confidence and improves the retention of ASL signs, ensuring an effective and continuous learning journey.

Stakeholders Feedback	
"When using the practice feature to cycle through the ASL alphabet and words, how seamless is your navigation experience? Are you able to move through the practice sets without any interruptions or difficulties?"	
Yusuf	*“Going through the letters and words feels really smooth. It helps a lot when I want to focus on a tricky letter or a new word I’ve just learnt. There was one time though when it seemed to stick for a moment before moving to the next sign, not sure if it was just a one-off glitch. Making sure that navigation is always this smooth would really help keep my focus on learning without getting interrupted by tech issues.”*
Robert	*“The navigation functionality is intuitive and responsive, which suits my fast-paced environment. The ability to quickly switch between letters and words is critical for my targeted practice sessions. On occasion, there is a slight lag, particularly when switching from letters to words, but it's not a frequent problem. An even more responsive system would ensure an absolutely seamless experience, but as it stands, it's already a highly useful tool in my daily practice.”*
Imtiaz	*“I've had no trouble moving between the different signs and words. It’s pretty straightforward!. The system reacts well to my input, which keeps the flow of learning going.”*

Stakeholders Yusuf, Robert, and Imtiaz share their experiences with the navigation feature within the ASL learning software, highlighting its smoothness and intuitiveness.

Yusuf emphasises a fairly fluid flow between letters and words, with one brief period of pause, emphasising the significance of consistent navigation for focused learning. This feedback suggests the potential need for optimising more state management within the software to handle transitions and user inputs without delays. This could involve caching the next expected inputs or using more efficient data structures to manage the state transitions.

Robert appreciates the intuitive and responsive navigation, essential for his fast-paced professional setting. He notes a slight lag when transitioning from letters to words, indicating that while the system is highly functional, there's room for increased responsiveness. For this, I could code an anticipatory loading feature that pre-emptively loads the next letters or words in the background while the user is still on the current one to ensure instant display upon request.

Imtiaz finds the navigation straightforward and has experienced no trouble, emphasising the system's responsiveness to his inputs. His feedback affirms the system's effective design in facilitating a continuous learning flow.

Evaluation

The evaluation of the project against the success criteria was thoroughly conducted using evidence gathered from Post-Development Testing.

Usability

For a program to be successful, it must not only satisfy the predefined success criteria but also offer a user-friendly experience. User satisfaction is essential for my project; if a program isn't intuitive and enjoyable to use, it risks being abandoned by users, even if it has the capabilities to meet their needs.

Usability testing was conducted by the stakeholders. Considering their varied skill sets as described in their description in the Stakeholders section, their collective feedback can be extrapolated to measure broader user satisfaction. Following a testing methodology similar to the one outlined in the Post-Development Testing section, stakeholders independently utilised the program and shared their insights through communication with email. This approach, free from structured testing constraints, aimed to capture their genuine experience with the program. Additionally, opting for individual email response instead of a group meeting ensured that stakeholders provided independent feedback, avoiding the potential influence of one participant's reactions on another's.

Met Success Criteria (Stakeholder Standpoint)

Confirming the program's high usability aligns with the established Success Criteria. Each criterion will be assessed based on the insights from the preceding discussion and the ratings (out of 5) collected from the Post-Development Testing section. Additionally, a brief comment summarising the findings will accompany the ratings for clarity.

Not met	Partially met	Fully met
Requires serious improvements	Requires improvement	Required minimal or no improvements

Success Criteria (User Requirement)	Description	Rating	Comments
Access user’s front camera	Users will see real-time feed from their front camera, providing immediate visual feedback for hand positioning and gesture recognition.		Stakeholders reported confusion during software use due to the front camera providing a non-mirrored view, where their right hand appeared on the left side of the screen. A mirrored view would align with their natural expectations and increase comfort. Adjusting to a mirrored display can be efficiently handled with OpenCV.
Bounding box	Users will observe a dynamically adjusting, bright bounding box around their hand, clarifying the focus area for gesture detection.		My stakeholders are very pleased with the hand detection feature, which consistently follows the hand in real-time without any delay. They have provided positive feedback, particularly appreciating its ability to facilitate gesture recognition.
Model accuracy	Users receive a percentage-based accuracy score, offering feedback on gesture recognition performance.		My stakeholders love the accuracy percentage feature as it provides immediate feedback. Enhancing this could involve using more images, giving the model more examples to train from.

Letter prediction	Users see the predicted letter displayed prominently, giving instant validation of the signed gesture.		This works extremely well, for this first prototype. However, in order to ensure users are fully satisfied, I need to train the model with a large amount of data for each letter. Obviously, this requires a vast amount of storage, so that model predicts the sign accurately.
Letter	Users focus on a large, central display of the current target letter, which guides the fingerspelling practice.		This feature captures the stakeholders' attention as it is prominently displayed in large font in the middle of the screen. Its central positioning ensures that it is immediately noticeable, facilitating easier interaction and engagement.
Skip letter function	Users can press a key to quickly skip letters, allowing personalised pacing and control over the learning session.		My stakeholders have really liked the software's transition feature between signs, which facilitates uninterrupted progressive learning. However, I overlooked the possibility that users might want to revisit previous words for additional practice, an improvement worth considering for future updates.
Word	Users are presented with the current word for fingerspelling practice, with the upcoming word visible.		My stakeholders appreciate this feature because it enables them to practise in real-world settings rather than focusing solely on individual letters. An improvement to consider is modifying the code to allow users to add their own words for personalised practice.
Skip word function	Users can bypass the current word, facilitating a tailored learning experience that adapts to user confidence and skill level.		My stakeholders also liked this feature for the same reasons as the skip letter function. Similarly, an enhancement to consider is adding a function that allows users to return to previous words for additional practice.
GUI background	Users experience a comforting, neutral-toned GUI background that minimises distractions and enhances readability.		The GUI performed as expected, and I received positive feedback from stakeholders. However, it differed from the initial design specified in the Design section because OpenCV could not support smaller font sizes. For future improvements, considering a different library for the GUI may be beneficial to enhance flexibility and design options.
Bounding box colour	Users find the hand's bounding box in a bright, attention-catching colour, promoting better visual tracking of hand positioning.		The colour of the bounding box worked well against dark backgrounds but tended to blend in with lighter ones. To enhance visibility, I may need to adjust the colour or, alternatively, allow users to customise the bounding box colour themselves, providing a more personalised learning experience.
Text colour	Users can easily read text due to the high-contrast deep purple hue set against the lighter GUI background, aiding focus and clarity.		The text colour contrasts effectively with the light colour of the GUI background, making the text easily readable for my stakeholders. This clear visibility enhances user interaction and overall satisfaction with the software.

Inform user to begin by placing hand in front of camera	Users are prompted to initiate interaction by a clear message, signalling when the system is ready for hand recognition.		My stakeholders really appreciated the initial prompt, "Place hand in front of the camera to begin," as it highlights the user-friendliness of the software, especially for newcomers. This clear instruction effectively guides first-time users, easing their introduction to the software.
Inform user hand is not localised	Users receive real-time prompts when hand placement is not detected, guiding them to reposition for successful recognition.		The message "Place hand in front of camera" was highly regarded by my stakeholders, informing them that the software requires their hand to be visible before proceeding with recognition. This feature effectively communicates the need for user interaction to initiate the process.
Inform user to position hand on the left hand side of the screen	Users are instructed to move their hand to the visible left side of the screen, ensuring clear sight and detection.		My stakeholders also expressed a strong preference for the message "Move hand on the left-hand side of the screen." To further enhance this feature, the visibility of these messages could be improved to make them more eye-catching to the user.
Inform user correct hand spelling	Users are immediately praised with positive feedback when signs are correctly identified, enhancing the learning experience.		When the messages “Well Done!” and “Correct!” are displayed for just 1 millisecond in the smallest font provided by OpenCV, they are not very noticeable to the user. Addressing these visibility issues will enhance the motivational impact of these messages, further engaging and encouraging the users.
X button	Users have an intuitive and accessible way to exit the program, ensuring they can effortlessly control.		My stakeholders reminded me of this issue where the window reopens immediately after pressing the X button to close it. To close it, they will have to stop running the program. This oversight can be swiftly addressed in the software's exit functionality.

Usability Features Review

Relating back to GUI Design and Usability Features, The designed GUI elements, intended to make the program more user-friendly, have mostly been included.

Usability Feature	Description	Rating	Comments
Colour Scheme	The beige background will provide a clear contrast to the deep purple used for letters and words, with instructional texts in a distinct lighter purple to guide users effectively. This colour palette enhances visibility and user focus, facilitating a clear and intuitive learning experience.		The colour palette I selected did not appear as intended; it appears overly simplistic. My goal was to create a calm and friendly look suitable for users of all ages, but the current design lacks the intended appeal. To enhance the visual appeal and align with my vision, I am considering incorporating additional design elements similar to those found on fingerspelling.xyz. Despite these issues, I have generally adhered to the predefined design specifications.

Bounding Box	The brightly coloured bounding box ensures users can easily see and position their hands for accurate gesture recognition.		Despite initially using a preset colour for the bounding box, which was expected to stand out against the background, I had to switch to a brighter purple as suggested by my stakeholders during development. However, upon further review, I've noticed that this colour is sometimes still not distinguishable enough. Therefore, I am considering adding a feature that allows users to customise the colour of the bounding box. This enhancement will not only improve visibility but also build trust with users by giving them greater control over the software.
Responsive Feedback	The software's instant, colour-coded feedback, like a green "Well Done" for correct signs, clearly signals user success and motivates continued learning.		Feedback is interactive, as wanted. Whenever the user performs an action, a corresponding feedback message is displayed. Text colour is dark enough to be distinguished from background colour. Colour of positive feedback had to be changed, as it was very bright, easily blended in with the background colour, as stakeholders suggested.
Font and Size	The software will use a simple font for readability, ensuring text is comfortably legible for all users.		The font size options offered by OpenCV are quite limited, which constrained my ability to make the current letter stand out. Ideally, the letter should be the first thing a user notices, allowing smaller instructions to remain less obtrusive. Unfortunately, achieving this visual hierarchy was challenging due to these limitations.
Logo	The logo contributes to aesthetics but the focus is on functionality affecting user experience.		I did not include a logo in the current version of the software as it is not the main focus at this time. However, I could consider adding a simple emoji, as suggested by one of my stakeholders, to enhance the visual appeal.

Robustness

The program has been developed with validation routines to handle various types of inputs, including normal, boundary, and invalid, to prevent syntax or logical errors. These validations were thoroughly tested for robustness during iterative development, fully meeting the sub-criteria of the established success criteria:

- **Continuous Video Capture:** The code uses a while `True` loop to continuously capture video frames from the webcam, ensuring that my software remains responsive and ready to detect and analyse hand gestures as long as the program is running.

- **Hand Detected:** The software utilises the `HandDetector` module from CVZone, which is designed to accurately detect hands within the video frame. This module provides reliable hand tracking, even if the hand moves quickly or changes orientation. The robustness is further ensured by setting the `maxHands`

parameter to 1, which helps the system focus solely on the primary hand in view, reducing complexity and potential errors from multiple hand detections.

- **Dynamic Bounding Box:** By integrating `cvzone.HandTrackingModule`, I enable dynamic detection of hands in the video feed. This module is particularly robust against various background conditions and is capable of accurately identifying hand positions and movements, crucial for real-time sign language recognition.

- **Bounding Box Adjustment:** The software dynamically adjusts the bounding box based on the hand's position and size, ensuring that the hand remains within the detectable area of the camera feed. The use of offset provides a margin around the detected hand, allowing for slight movements without losing the hand from the bounding area. This is crucial for maintaining continuous gesture recognition even when the user moves their hand.

- **Error Detection for Non-Detection:** The software handles scenarios where no hands are detected by resetting the model accuracy to zero. This prevents my software from acting on incorrect or stale data, ensuring that actions are based only on current and accurate detections.

- **Image Aspect Ratio:** The code handles different aspect ratios of the hand by adjusting the resizing parameters. This ensures that the hand's image is appropriately scaled to fit the input size expected by the classification model, maintaining the accuracy of gesture recognition across different hand sizes and shapes.

- **Letters/Words Transition:** The system intelligently handles transitions between letters and words, automatically advancing to the next letter or word upon successful recognition. This automated flow minimises user intervention and enhances the learning experience by providing a smooth transition. Additionally, it prevents the system from being stuck on a single letter or word, thus avoiding frustration from the user's end.

- **Protection Against Infinite Loops:** The implementation of skipping letters and words through keyboard inputs ('l' and 'w') prevents users from getting stuck on a particular sign or word. It also gives them control over the pace of their learning, enhancing user interaction and engagement.

Maintainability

To ensure my code remains maintainable, I've integrated several features and practices that facilitate ease of understanding, modification, and extension by other developers.

- **Modular Design:** My code is structured in modular blocks, separating functionalities like hand detection, gesture recognition, and user interface management. This design allows developers to work on or improve individual components without affecting others, making the system easier to manage and debug.

- **Comments:** Throughout the code, I've included comprehensive comments explaining the purpose of functions and key sections. This will assist new developers in quickly understanding the project's mechanics.

- **Consistent Naming Conventions:** I've used clear and consistent naming conventions for variables and functions, which adhere to Python standards. This consistency ensures that any developer familiar with Python can easily understand what each part of the code is intended to do.

- **Use of Configurable Files:** Configuration settings such as model paths and UI preferences are stored in external files, allowing non-programmers or other developers to adjust the software's behaviour without diving into the core code. This separation of configuration and code enhances flexibility and customisation.

- **Parameterisation:** Functions are designed to accept parameters where possible, promoting reuse and reducing redundancy. This approach allows changes to be made in one place to affect multiple functionalities, a principle that significantly simplifies maintenance and updates.

- **Constants for Configuration:** By defining constants such as `offset` and `imgSize` at the top of the script, I make it easy for other developers to configure and tweak the software's settings. These constants act as parameters that can be adjusted without digging through the deeper logic of the code.

- **Code Segmentations:** By logically segmenting the code into distinct sections for initialisation, main execution loops, and utilities, I've ensured that developers can easily locate and understand specific parts of the program without needing to comprehend it entirely at once.

Solution Limitations

The Fingerspelling Recognition Software was successfully developed, however there are some limitations when compared to initial design and user expectations. I am going to first address the limitations that my stakeholders found, during the Post-Development Testing, and provide an ideal solution. Then I am going to address the limitations that I found when testing it myself, and with other users.

Stakeholders' Limitations Solutions

I am going to address only the main limitations, as some of the smaller ones I have already discussed during the review of my stakeholders' response.

1. **Lighting Conditions**: Robert's observation concerning the delay in dimmer conditions suggests a restriction in the software's capacity to respond to different lighting scenarios. This issue can be addressed by optimising the code to automatically alter camera settings based on ambient light. The method consists in building a dynamic exposure adjustment algorithm that can detect the average brightness of taken photos and compare them to a predetermined threshold. If the brightness falls below this threshold, the system can either alter the camera's exposure or show the user a visual prompt to improve the lighting circumstances:

```
import cv2

def adjust_camera_exposure(frame, target_brightness=130):
    # Convert to grayscale for brightness analysis
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    # Calculate the average brightness of the frame
    average_brightness = np.mean(gray)

    # Check if the brightness is below the target threshold
    if average_brightness < target_brightness:
          # Increase the exposure by reducing the 'brightness' setting of the camera
        # This would be specific to the camera being used and may involve
        # sending commands to the camera driver or using proprietary SDK functions.
        pass  # Implement camera-specific exposure adjustment here

    return frame

cap = cv2.VideoCapture(0)

while True:
    success, img = cap.read()
    img = adjust_camera_exposure(img)  # Adjust the camera exposure dynamically
    imgOutput = img.copy()
    hands, img = detector.findHands(img)
```

In the provided `adjust_camera_exposure` function, we first convert the frame to grayscale, which simplifies the process of calculating the average brightness. We then use numpy to compute the mean brightness of the image. If the brightness is below a set target, we attempt to increase the exposure. The specific method of increasing the exposure would depend on the camera's capabilities and APIs.

2. **Webcam Variation**: Imtiaz's experience points to a common issue where the performance of the software is dependent on the quality of the webcam being used. The variation in response times can be due to differences in frame rate, resolution, and latency inherent in older or lower-quality webcams. To address this, a calibration feature can be implemented. This feature would initially assess the webcam's frame rate and resolution, and then adjust the software settings to optimise performance. The calibration tool would perform a series of quick tests to evaluate the camera's capabilities, like capturing a burst of images to measure the actual frames per second and checking for motion blur to measure the camera's responsiveness:

```
import cv2
import time

def calibrate_camera(cap):
    # Check the maximum resolution the camera can handle
    cap.set(cv2.CAP_PROP_FRAME_WIDTH, 1920)
    cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 1080)

    # Check if the camera supports HD resolution
    if cap.get(cv2.CAP_PROP_FRAME_WIDTH) < 1920:
        # If not, set to a lower resolution
        cap.set(cv2.CAP_PROP_FRAME_WIDTH, 640)
        cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)

    # Test the camera's frame rate
    start_time = time.time()
    frames_tested = 0
    while frames_tested < 60:
        ret, frame = cap.read()
        if ret:
            frames_tested += 1
    end_time = time.time()

    # Calculate frames per second (FPS)
    fps = frames_tested / (end_time - start_time)
    print(f"Camera FPS: {fps:.2f}")

    # Based on the FPS, adjust the software's processing rate
    # This is a simplistic approach, more nuanced adjustments may be needed
    if fps < 30:
        # Set the software to process at a rate suitable for lower FPS
        pass  # Implement adjustments accordingly

cap = cv2.VideoCapture(0)
calibrate_camera(cap)

while True:
    # Main loop for the application
    pass
```

This code segment starts by trying to set the camera to its highest possible resolution and then checks to see if the camera supports it. If the camera does not support HD, it defaults to a standard resolution. It then captures 60 frames to measure the actual FPS. This information can be used to adjust the processing

algorithms of the software, such as reducing the complexity of image processing tasks or adjusting the detection algorithm to accommodate the lower frame rate.

3. **Training Model**: Training the model is indeed a crucial step that determines the effectiveness of any machine learning application. While 300 images per letter might seem sufficient, diversity in the dataset is key for the model to learn various representations of the same sign. This includes variations in background, hand orientation, skin tone, and other environmental factors. Incorporating a more diverse dataset helps prevent overfitting, which seems to have occurred here; the model performs well when tested with my hand but struggles with different users.

   For the CNN architecture, if the current setup is not capturing the details of the hand gestures effectively, enhancing the model's complexity might be beneficial. Increasing the depth (more layers) and breadth (more neurons) of the network can allow the model to learn more abstract representations of the data.

   Here's a possible update to the CNN architecture to make it more robust:

```
# Updated CNN architecture
model = Sequential([
   Conv2D(64, (3, 3), activation='relu', input_shape=(300, 300, 3)),
   MaxPooling2D(2, 2),
   Conv2D(128, (3, 3), activation='relu'),
   MaxPooling2D(2, 2),
   Conv2D(256, (3, 3), activation='relu'),
   MaxPooling2D(2, 2),
   Conv2D(256, (3, 3), activation='relu'),
   MaxPooling2D(2, 2),
   Flatten(),
   Dense(1024, activation='relu'),
   Dropout(0.5),
   Dense(24, activation='softmax')  #24 classes (A-Y excluding J and Z)
])
```

   This updated architecture introduces additional convolutional layers with more filters, which can detect more complex patterns in the images. The fully connected layer's size has also been increased, providing a larger model capacity to learn from the data.

   For hyperparameters, tweaking them could lead to better results. Here's an example:

```
# Hyperparameters update
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

# New hyperparameters for training
history = model.fit(
   train_generator,
   steps_per_epoch=200,  # Increased for more training iterations per epoch
   epochs=25,  # More epochs for extended training
   validation_data=validation_generator,
   validation_steps=100,  # Increased for more thorough validation
   verbose=2)
```

   In the updated code, the number of epochs and the steps per epoch for both training and validation have been increased. This will allow the model to see the data more times and learn more from it.

   Lastly, implementing more data augmentation methods such as random rotations, shifts, zooming, and

flipping can further enhance the diversity of your dataset without needing to collect new data. This can be integrated into the `ImageDataGenerator`. I will have to remember to evaluate the changes with a validation set to ensure improvements are actual and not a result of overfitting.

4. **Visual Cues:** To meet Yusuf and Imtiaz's requests for more nuanced feedback and assistance, incorporating an intelligent feedback system into the fingerspelling software could significantly improve the learning experience. When the model's accuracy goes below 100%, it is critical to provide the user with detailed feedback on how to improve their motions. This can include noting the specific element of the gesture that requires adjustment, such as finger location or movement speed.

   A real-time mistake correction guide that modifies dynamically based on the user's performance might be used to do this. This guide would use visual cues, like arrows or highlights, on the video feed to direct the user's hand movements more accurately. The example below shows how this could be integrated into the existing system using OpenCV for image annotations and conditions based on model feedback:

```
import cv2
import numpy as np

def guide_user_correction(imgOutput, prediction, labels, index,
accuracy_threshold=0.9):
    # If the accuracy is below threshold and the prediction is incorrect
    if prediction[index] < accuracy_threshold:
        text = f"Adjust: {labels[index]}"
        # Display corrective suggestion
        cv2.putText(imgOutput, text, (10, 50), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255,
0), 2, cv2.LINE_AA)
        # Example of drawing an arrow for correction
        cv2.arrowedLine(imgOutput, (300, 300), (300, 200), (0, 255, 0), 3)

while True:
    success, img = cap.read()
    img = cv2.flip(img, 1)  # Adjust to mirror the image
    imgOutput = img.copy()
    hands, img = detector.findHands(img)
    if hands:
        hand = hands[0]
        bbox = hand['bbox']
        imgCrop = img[bbox[1]:bbox[1]+bbox[3], bbox[0]:bbox[0]+bbox[2]]
        prediction, index = get_prediction(imgCrop)
        guide_user_correction(imgOutput, prediction, labels, index)

    cv2.imshow("Fingerspelling Recognition Software", imgOutput)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
```

In this code, `guide_user_correction` is a function that overlays instructional graphics on the `imgOutput` based on the current prediction's accuracy. If the accuracy for the current sign falls below a certain threshold, it displays text and an arrow to guide the user on how to correct their sign. The arrow's direction and position can be adjusted to reflect specific guidance, such as "move hand higher" or "closer together fingers."

This system could be further enhanced by using more sophisticated methods to analyse the specific parts of the hand that are incorrectly positioned, utilising advanced computer vision techniques or more machine learning models trained to recognise correct and incorrect positioning in more granular detail. This would make the feedback not only reactive but also proactive, helping users correct their signs before making mistakes.

5. **Animations:** To enrich the user experience during transitions between letters and words, incorporating Yusuf's and Robert's suggestions could significantly elevate user engagement and learning efficiency. Enhancing message interactivity and visibility will ensure users are not only aware of the transitions but also understand them intuitively.

   Implementing dynamic visual cues such as flashing text, colour changes, or animated arrows could guide users more clearly through the learning process. These enhancements help maintain user attention and reinforce the learning objectives without overwhelming the user. Here is an example of how you might modify the software to incorporate these features:

```
import cv2

def display_transition_effects(imgOutput, text, position, effect='flash', color=(0,
255, 0)):
    if effect == 'flash':
        # Flashing effect by alternating the visibility of the text
        if int(cv2.getTickCount() % 1000) < 500:
            cv2.putText(imgOutput, text, position, cv2.FONT_HERSHEY_SIMPLEX, 1,
color, 2)
    elif effect == 'highlight':
        # Draw a highlighted rectangle around text
        text_size = cv2.getTextSize(text, cv2.FONT_HERSHEY_SIMPLEX, 1, 2)[0]
        top_left = (position[0] - 5, position[1] - text_size[1] - 5)
        bottom_right = (position[0] + text_size[0] + 5, position[1] + 5)
        cv2.rectangle(imgOutput, top_left, bottom_right, color, cv2.FILLED)
        cv2.putText(imgOutput, text, position, cv2.FONT_HERSHEY_SIMPLEX, 1, (255,
255, 255), 2)

while True:
    success, img = cap.read()
    img = cv2.flip(img, 1)
    imgOutput = img.copy()
    hands, img = detector.findHands(img)

    # Simulate transition to new content
    display_transition_effects(imgOutput, "Next Letter: B", (50, 50),
effect='flash', color=(255, 0, 0))
    display_transition_effects(imgOutput, "Correct!", (50, 100), effect='highlight',
color=(0, 255, 0))

    cv2.imshow("Fingerspelling Recognition Software", imgOutput)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
```

   In this revised code, the `display_transition_effects` method can be used to add various visual effects. For example, it can flash text to grab attention or highlight text to emphasise the current action required. This interactivity not only aids in maintaining engagement but also enhances the clarity of instructional content, helping users navigate through the learning material more effectively.

   Adding such animated and visually stimulating elements can make the interface more intuitive and responsive, catering to both beginners and advanced learners by providing them with a clear path to follow, thus enhancing the overall educational impact of the software.

General Limitations Solutions

During the Post-Development Testing phase, I identified several limitations in the software that I anticipated would not meet my stakeholders' approval. Surprisingly, some of these issues were not highlighted by them. I plan to address each of these concerns and outline potential solutions to maximise user experience.

1. **Camera View**: The default view of the OpenCV live feed is not mirrored but inverted, which I initially did not consider as an issue. However, during testing with volunteers, many found it challenging to adapt to this orientation, mistakenly raising their left hand when the camera frame appeared on the left side of the screen. This confusion stems from the natural expectation of seeing a mirror image of oneself, similar to a reflection. The frustration from this unexpected behaviour was evident among testers unaccustomed to the inversion. Fortunately, this can be easily fixed, as OpenCV allows the camera feed to be flipped to any desired orientation.

   To display a mirrored image of the camera feed in my application, I need to flip the image horizontally. This can be done by adding a single line of code to flip the frame captured from the camera:

   ```
   while True:
       success, img = cap.read()  # Capture frame from camera
       img = cv2.flip(img, 1)  # Flip the frame horizontally to get a mirror image
       imgOutput = img.copy()  # Main display
       hands, img = detector.findHands(img)  # Detect hands and annotate the frame
       ...
   ```

   In the `cv2.flip()` function, the `1` indicates that the image should be flipped around the y-axis, which mirrors the image horizontally. This adjustment ensures that when users raise their right hand, it appears on the right side of the screen, making the interaction more intuitive and mirroring a true reflection.

2. **Customisable Settings:** Introducing customisable settings into the fingerspelling software significantly enhances user engagement by allowing them to tailor the learning experience to their preferences. This not only increases trust in the software's capabilities but also addresses individual learning styles and environmental variations.

   a. **Customisable Accuracy Threshold:** Users can set their own accuracy thresholds for progressing to the next letter, which offers a more personalised and challenging learning curve. Currently, the transition to the next letter is based on the highest accuracy among the other letters. By enabling users to define what constitutes sufficient accuracy, learners can ensure they are truly mastering each sign before moving on.

   ```
   # User-defined accuracy threshold
   user_accuracy_threshold = 0.90  # 90% accuracy

   def check_accuracy(prediction, index, threshold):
       if prediction[index] >= threshold:
           return True
       return False

   if check_accuracy(prediction, index, user_accuracy_threshold):
       # Move to the next letter
       ...
   ```

   In this code, the `check_accuracy` function compares the prediction accuracy of the current letter against a user-defined threshold. If the prediction accuracy meets or exceeds this threshold, the software progresses to the next letter. This method allows users to challenge themselves by setting higher thresholds or adjust for easier progression if they are beginners.

b. **Customisable Bounding Box Colour:** Given the challenges with visibility of the bounding box against various backgrounds, allowing users to select their own bounding box colour can greatly enhance usability. This customisation helps ensure that the bounding box is always visible, regardless of the background or lighting conditions.

```
# Function to change bounding box colour based on user preference
def update_bounding_box_colour(img, bbox, colour):
    x, y, w, h = bbox
    cv2.rectangle(img, (x, y), (x+w, y+h), colour, 2)

# User selects a colour
user_selected_color = (255, 0, 255)  # Bright magenta

if hands:
    hand = hands[0]
    bbox = hand['bbox']
    update_bounding_box_color(imgOutput, bbox, user_selected_color)
```

This code allows the bounding box colour to be dynamically updated based on the user's selection. The function `update_bounding_box_colour` takes the image, bounding box coordinates, and the user-selected colour as parameters, drawing the bounding box in the chosen colour. This flexibility helps users maintain visibility of the hand detection area, thereby improving their ability to correctly position their hands for gesture recognition.

References

[1] M. MacSweeney, “Fingerspelling, signed language, text and picture processing in deaf native signers: The role of the mid-fusiform gyrus,” ncbi.nlm.nih.gov, Apr 2007. [Online]. Available at: www.ncbi.nlm.nih.gov/pmc/articles/PMC3480647 [Last Accessed: 4 July 2023].

[2] M. G. Suraj, D. S. Guru, “Appearance based Recognition Methodology for Recognising Fingerspelling Alphabets,” ijcai.org, 2007. [Online]. Available at: www.ijcai.org/Proceedings/07/Papers/096.pdf [Last Accessed: 28 September 2023].

[3] R. S. Aouf, “Fingerspelling.xyz app helps people to learn the sign language alphabet,” dezeen.com, Aug 2021. [Online]. Available at: www.dezeen.com/2021/08/27/fingerspelling-xyz-app-learn-sign-language-alphabet-design [Last Accessed: 30 September 2023].

[4] HELLOMONDAY/DEPT® “‘Fingerspelling’ Campaign by American Society for Deaf Children Empowers Parents of Deaf Children,” campaignsoftheworld.com, Jun 2023. [Online]. Available at: www.campaignsoftheworld.com/technology/fingerspelling [Last Accessed: 30 September 2023].

[5] M. Barrett “Ace ASL (American Sign Language) Review – Is the App Worth it?,” edwize.org, May 2023. [Online]. Available at: www.edwize.org/ace-asl [Last Accessed: 13 October 2023].

[6] T. L. A Khoa, “Ace ASL APK For Android Download,” m.apkpure.com, Jul 2022. [Online]. Available at: www.m.apkpure.com/ace-asl-learn-fingerspelling/us.signall.learnfingerspell [Last Accessed: 5 October 2023].

[7] Ultraleap, “Small. Fast. Accurate. World-class hand tracking for anyone, anywhere,” ultraleap.com, 2023. [Online] Available at: https://www.ultraleap.com/leap-motion-controller-overview [Last Accessed: 25 October 2023].

[8] T. W. Chong, B. G. Lee, “American Sign Language Recognition Using Leap Motion Controller with Machine Learning Approach,” mdpi.com, Oct 2018. [Online]. Available at: www.mdpi.com/1424-8220/18/10/3554 [Last Accessed: 25 October 2023].

[9] O. Domanska, “Machine Learning vs Traditional Programming,” avenga.com, Dec 2021. [Online]. Available at: https://www.avenga.com/magazine/machine-learning-programming [Last Accessed: 30 December 2023].

[10] J. Brownlee “A Gentle Introduction to Object Recognition With Deep Learning,” machinelearningmastery.com, Jan 2021. [Online]. Available at: https://machinelearningmastery.com/object-recognition-with-deep-learning [Last Accessed: 5 January 2024].

[11] H. Thilakarathne “Using Hierarchical Data Format (HDF5) in Machine Learning,” naadispeaks.blog, Jun 2021. [Online]. Available at: https://naadispeaks.blog/2021/06/30/using-hierarchical-data-format-hdf5-in-machine-learning [Last Accessed: 15 February 2024].

[12] “Build from source,” tensorflow.org, Dec 2023. [Online]. Available at: https://www.tensorflow.org/install/source [Last Accessed: 29 February 2024].

[13] “On Common Split for Training, Validation, and Test Sets in Machine Learning,” medium.com, Mar 2019. [Online]. Available at: https://pub.towardsai.net/breaking-the-mold-challenging-the-common-split-for-training-validation-and-test-sets-in-machine-271fd405493d [Last Accessed: 1 March 2024].

[14] “Why You Need to Compile Your Keras Model Before Using model.evaluate(),” saturncloud.io, Jul 2023. [Online]. Available at: https://saturncloud.io/blog/why-you-need-to-compile-your-keras-model-before-using-modelevaluate/ [Last Accessed: 10 March 2024].

[15] "Keras MaxPooling2D," educba.com, March 2023. [Online]. Available at: https://www.educba.com/keras-maxpooling2d/ [Last Accessed: 19 March 2024].

Code Appendix

Folder Hierarchy Diagram

Data/

```
│
├── train/
│    ├── A/
│    ├── B/
│    ├── C/
│   ...
│    ├── X/
│    └── Y/
│
├── validation/
│    ├── A/
│    ├── B/
│    ├── C/
│   ...
│    ├── X/
│    └── Y/
│
└── test/
     ├── A/
     ├── B/
     ├── C/
    ...
     ├── X/
     └── Y/
```

Model/

labels.txt

model.h5

words.txt

dataCollection.py

```
import cv2
from cvzone.HandTrackingModule import HandDetector
import numpy as np
import math
import time

cap = cv2.VideoCapture(0)  # Initialise video capture
detector = HandDetector(maxHands=1)  # Initialise HandDetector object

# Constants
offset = 20  # Expand bounding box
imgSize = 300  # Image dimensions
maxImg = 300  # Maximum number of images
counter = 0

letters = ["A", "B", "C", "D", "E", "F", "G", "H", "I", "K", "L", "M", "N", "O",
"P", "Q", "R", "S", "T", "U", "V", "W", "X", "Y"]
letters_index = 0

while True:
    success, img = cap.read()  # Capture frame from camera
    hands, img = detector.findHands(img)  # Detect hands and annotate the frame

    if hands:
        hand = hands[0]
        x, y, w, h = hand['bbox']  # Getting bounding box information
        imgCrop = img[y-offset:y+h+offset, x-offset:x+w+offset]
        imgWhite = np.ones((imgSize, imgSize, 3), np.uint8) * 255  # Image
background

        imgCropShape = imgCrop.shape

        # Image fitting
        aspectRatio = h / w

        # Height is greater
        if aspectRatio > 1:
            k = imgSize / h
            wCal = math.ceil(k * w)
            imgResize = cv2.resize(imgCrop, (wCal, imgSize))  # Resize imgCrop
            #imgResizeShape = imgResize.shape  # Retrieve shape of imgResize
            wGap = math.ceil((imgSize - wCal) / 2)  # Used to centre imgCrop
            imgWhite[:, wGap:wCal+wGap] = imgResize

        # Width is greater
        else:
            k = imgSize / w
            hCal = math.ceil(k * h)
            imgResize = cv2.resize(imgCrop, (imgSize, hCal))  # Resize imgCrop
```

```
            #imgResizeShape = imgResize.shape
            hGap = math.ceil((imgSize - hCal) / 2)  # Used to centre imgCrop
            imgWhite[hGap:hCal + hGap, :] = imgResize

        cv2.imshow("ImageCrop", imgCrop)
        cv2.imshow("ImageWhite", imgWhite)

    cv2.imshow("Image", img)

    key = cv2.waitKey(1)

    if key == ord("s"):  # Save imgWhite
        counter += 1  # Increment number of images taken
        if counter > maxImg:
            print((counter % maxImg), "Press R for next letter.")  # maxImg
reached
        else:
            folder = "Data/" + letters[letters_index]  # Current directory
            cv2.imwrite(f'{folder}/Image_{time.time()}.jpg', imgWhite)  # Save
imgWhite
            print(letters[letters_index], counter)  # Current letter

    if key == ord("r"):  # Change folder
        counter = 0  # Reset number of images taken
        letters_index += 1  # Move onto next letter
        print("Folder changed to", letters[letters_index])
```

folderSort.py

```
import os
import shutil

mainDirectory = "Data/"  # Base directory path
subDirectories = ["train", "validation", "test"]
letters = ["A", "B", "C", "D", "E", "F", "G", "H", "I", "K", "L", "M", "N", "O",
"P", "Q", "R", "S", "T", "U", "V", "W", "X", "Y"]

# Create train, validation, and test directories
for subDirectory in subDirectories:
    for letter in letters:
        dir_path = os.path.join(mainDirectory, subDirectory, letter)  # Construct
path
        if not os.path.exists(dir_path):  # If path doesn't exist, create it
            os.makedirs(dir_path)

# Distribute images into train, validation, and test sets
def split_data(letter, images):
    numImg = len(images)
    # Calculate splits (limits):
    trainSplit = int(0.8 * numImg)  # (80% train)
    validationSplit = int(0.9 * numImg)  # (10% validation)
    # Test split is implicitly the remaining images  # (10% test)

    # Split the images
    trainImages = images[:trainSplit]
    validationImages = images[trainSplit:validationSplit]
    testImages = images[validationSplit:]

    # Move images to directory
    def move_images(images, destinationDirectory):
        for img in images:
            shutil.move(img, os.path.join(mainDirectory, destinationDirectory,
letter))

    # Move images to respective directories
    move_images(trainImages, "train")
    move_images(validationImages, "validation")
    move_images(testImages, "test")


# Iterate over each letter and process the images
for letter in letters:
    letterPath = os.path.join(mainDirectory, letter)
    if os.path.exists(letterPath):  # Check if the letter directory exists
        # List all files in the letter directory and get their full paths
        images = [os.path.join(letterPath, img) for img in os.listdir(letterPath)
if
                  os.path.isfile(os.path.join(letterPath, img))]
```

```
        split_data(letter, images)
        shutil.rmtree(letterPath)  # Remove the original path after moving all
 the images

print("Done")
```

main.py

```
import cv2
from cvzone.HandTrackingModule import HandDetector
from cvzone.ClassificationModule import Classifier
import numpy as np
import math
import random

cap = cv2.VideoCapture(0)  # Initialise video capture
detector = HandDetector(maxHands=1)  # Initialise HandDetector object
classifier = Classifier("Model/model.h5", "Model/labels.txt")

# Constants
offset = 20  # Expand bounding box
imgSize = 300  # Image dimensions

# Words
labels = ["A", "B", "C", "D", "E", "F", "G", "H", "I", "K", "L", "M", "N", "O",
"P", "Q", "R", "S", "T", "U", "V", "W",
         "X", "Y"]
wordsFile = "words.txt"

modelAccuracy = 0
currentWordIndex = 0
currentLetterIndex = 0
instructionSetConstant = 0

# Functions
def get_prediction(imgWhite):  # Retrieve model prediction, and index
    modelPrediction, modelIndex = classifier.getPrediction(imgWhite)
    return modelPrediction, modelIndex


def words_list(wordsFile):  # Put words into a list
    wordsList = []
    with open(wordsFile) as file:
        for line in file:
            wordsList.append(line.strip())
    return wordsList


def last_word(currentWordIndex, words):
    if currentWordIndex == len(words):  # Check if there are no words left
        currentWordIndex = 0  # Go back to initial word
    return currentWordIndex


while True:
    success, img = cap.read()  # Capture frame from camera
    imgOutput = img.copy()  # Main display
```

```
    hands, img = detector.findHands(img)  # Detect hands and annotate the frame

    # ASL Setup
    words = words_list(wordsFile)  # Place all words into a list
    currentWord = list(words[currentWordIndex])  # Split currentWord into
individual letters
    currentLetter = currentWord[currentLetterIndex]

    if hands:
        hand = hands[0]
        x, y, w, h = hand['bbox']  # Getting bounding box information
        imgCrop = img[y - offset:y + h + offset, x - offset:x + w + offset]
        imgWhite = np.ones((imgSize, imgSize, 3), np.uint8) * 255  # Image
background

        imgCropShape = imgCrop.shape

        # Image fitting
        aspectRatio = h / w

        if (0 < (x - offset) < 320) and ((y - offset) > 0):  # Prevents error
when x/y coordinates are negative

            # Height is greater
            if aspectRatio > 1:
                k = imgSize / h
                wCal = math.ceil(k * w)
                imgResize = cv2.resize(imgCrop, (wCal, imgSize))  # Resize
imgCrop
                # imgResizeShape = imgResize.shape  # Retrieve shape of imgResize
                wGap = math.ceil((imgSize - wCal) / 2)  # Used to centre imgCrop
                imgWhite[:, wGap:wCal + wGap] = imgResize
                prediction, index = get_prediction(imgWhite)

                # Bounding box
                cv2.rectangle(imgOutput, (x - offset - 2, y - offset - 30), (x +
w + 22, y - offset), (255, 0, 255),
                              cv2.FILLED)
                cv2.putText(imgOutput, labels[index], (x - 17, y - 23),
cv2.FONT_HERSHEY_PLAIN, 2, (255, 255, 255), 2)
                cv2.rectangle(imgOutput, (x - offset, y - offset), (x + w +
offset, y + h + offset), (255, 0, 255), 3)

            # Width is greater
            else:
                k = imgSize / w
                hCal = math.ceil(k * h)
                imgResize = cv2.resize(imgCrop, (imgSize, hCal))  # Resize
imgCrop
                # imgResizeShape = imgResize.shape
                hGap = math.ceil((imgSize - hCal) / 2)  # Used to centre imgCrop
```

```
                imgWhite[hGap:hCal + hGap, :] = imgResize
                prediction, index = get_prediction(imgWhite)

                # Bounding box
                cv2.rectangle(imgOutput, (x - offset - 2, y - offset - 30), (x +
w + 22, y - offset), (255, 0, 255),
                              cv2.FILLED)
                cv2.putText(imgOutput, labels[index], (x - 17, y - 23),
cv2.FONT_HERSHEY_PLAIN, 2, (255, 255, 255), 2)
                cv2.rectangle(imgOutput, (x - offset, y - offset), (x + w +
offset, y + h + offset), (255, 0, 255), 3)

            modelAccuracy = prediction[index]

        '''
        print(prediction[index], labels[index])
        '''

        # ASL Alphabet Learning
        letter = labels[index]  # Model letter prediction

        print(words[currentWordIndex], currentWordIndex)
        print(letter, currentLetter, currentLetterIndex)

        if currentLetter == letter:  # Check if model's prediction matches with
currentLetter
            if currentLetterIndex >= (len(currentWord) - 1):  # Check if
currentLetter reaches end of currentWord
                currentWordIndex += 1  # Move onto next word in the list
                currentLetterIndex = 0  # First letter of new word
                currentWordIndex = last_word(currentWordIndex, words)
            else:
                currentLetterIndex += 1  # Move onto next letter in currentWord

        if ((x - offset) > 320) or ((x - offset) < 0) or ((y - offset) < 0):  #
Hand is not visible
            modelAccuracy = 0

    else:
        modelAccuracy = 0

    # Model Accuracy
    cv2.putText(imgOutput, str(modelAccuracy * 100)[:4] + "%", (20, 40),
cv2.FONT_HERSHEY_PLAIN, 2, (255, 0, 255), 2)

    # GUI
    cv2.rectangle(imgOutput, (400, 0), (1000, 1000), (179, 227, 255), cv2.FILLED)
# Background
    cv2.putText(imgOutput, currentLetter, (485, 153), cv2.FONT_HERSHEY_PLAIN, 7,
(146, 32, 117), 7)  # currentLetter
    cv2.putText(imgOutput, words[currentWordIndex], (419, 280),
```

```
cv2.FONT_HERSHEY_PLAIN, 4, (146, 32, 117),
                4)  # currentWord
    cv2.putText(imgOutput, "Next word: " + words[last_word(currentWordIndex + 1,
words)], (442, 300),
                cv2.FONT_HERSHEY_PLAIN, 1, (146, 32, 117), 1)  # Next word

    # Skip Instructions
    cv2.putText(imgOutput, "Press (L) to move", (444, 175),
cv2.FONT_HERSHEY_PLAIN, 1, (51, 3, 41), 1)
    cv2.putText(imgOutput, "onto next letter.", (450, 195),
cv2.FONT_HERSHEY_PLAIN, 1, (51, 3, 41), 1)
    cv2.putText(imgOutput, "Press (W) to move", (440, 330),
cv2.FONT_HERSHEY_PLAIN, 1, (51, 3, 41), 1)
    cv2.putText(imgOutput, "onto next word.", (450, 350), cv2.FONT_HERSHEY_PLAIN,
1, (51, 3, 41), 1)

    # Instructions
    if len(hands) == 0 and instructionSetConstant == 0:  # Display initial
message
        cv2.putText(imgOutput, "Place hand in front of", (425, 415),
cv2.FONT_HERSHEY_PLAIN, 1, (0, 0, 255), 1)
        cv2.putText(imgOutput, "camera to begin.", (450, 435),
cv2.FONT_HERSHEY_PLAIN, 1, (0, 0, 255), 1)
    elif len(hands) == 0:  # Display message if no hands are detected
        cv2.putText(imgOutput, "Place hand in front", (434, 415),
cv2.FONT_HERSHEY_PLAIN, 1, (0, 0, 255), 1)
        cv2.putText(imgOutput, "of camera.", (475, 435), cv2.FONT_HERSHEY_PLAIN,
1, (0, 0, 255), 1)
    elif hands:
        instructionSetConstant = 1  # Change instructionSetConstant so that
initial message can no longer be displayed
        hand = hands[0]
        x, y, w, h = hand['bbox']  # Getting bounding box information
        if (x - offset) > 320:  # Inform user to move hand to the left
            cv2.putText(imgOutput, "Move hand on the left", (425, 415),
cv2.FONT_HERSHEY_PLAIN, 1, (0, 0, 255), 1)
            cv2.putText(imgOutput, "hand side of the screen.", (415, 435),
cv2.FONT_HERSHEY_PLAIN, 1, (0, 0, 255), 1)
        prediction, index = get_prediction(imgWhite)
        if currentLetter == labels[index]:
            randomMessage = random.randint(0, 1)  # Display random message
            if randomMessage == 1:
                cv2.putText(imgOutput, "Well Done!", (477, 425),
cv2.FONT_HERSHEY_PLAIN, 1, (0, 255, 0), 1)
            else:
                cv2.putText(imgOutput, "Correct!", (480, 425),
cv2.FONT_HERSHEY_PLAIN, 1, (0, 255, 0), 1)

    cv2.imshow("Fingerspelling Recognition Software", imgOutput)

    key = cv2.waitKey(1)
```

```
    # Skip Letter
    if key == ord("l"):
        print(currentLetter, "is skipped.")
        if currentLetterIndex >= (len(currentWord) - 1):  # Check if
 currentLetter reaches end of currentWord
            currentWordIndex += 1  # Move onto next word in the list
            currentLetterIndex = 0  # First letter of new word
            currentWordIndex = last_word(currentWordIndex, words)
        else:
            currentLetterIndex += 1  # Increment Letter

    # Skip Word
    if key == ord("w"):
        print(words[currentWordIndex], "is skipped.")
        currentLetterIndex = 0  # First letter of new word
        currentWordIndex = last_word(currentWordIndex + 1, words)
```

trainClassifier.py

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,
Dropout
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Define directory paths
train_dir = 'Data/train'
validation_dir = 'Data/validation'
test_dir = 'Data/test'

# Preparing the data
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest')

validation_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(300, 300),
    batch_size=20,
    class_mode='categorical')

validation_generator = validation_datagen.flow_from_directory(
    validation_dir,
    target_size=(300, 300),
    batch_size=20,
    class_mode='categorical')

# Building the model
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(300, 300, 3)),
    MaxPooling2D(2, 2),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Flatten(),
    Dense(512, activation='relu'),
    Dropout(0.5),
    Dense(24, activation='softmax')  # Assuming 24 classes (A-Y excluding J and
Z)
])
```

```
# Compiling the model
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

# Training the model
history = model.fit(
    train_generator,
    steps_per_epoch=100,  # Adjust based on your dataset size
    epochs=15,
    validation_data=validation_generator,
    validation_steps=50,  # Adjust based on your dataset size
    verbose=2)

# Evaluating the model
test_datagen = ImageDataGenerator(rescale=1./255)
test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=(300, 300),
    batch_size=20,
    class_mode='categorical')

eval_result = model.evaluate(test_generator)
print(f"\nTest Loss: {eval_result[0]}, Test Accuracy: {eval_result[1]}")

# Save the model
model.save('model.h5')
print("Model saved as model.h5")
```