

# **ASL - ALPHABET IMAGE RECOGNITION PROJECT REPORT**

## **1. INTRODUCTION**

- 1.1 Project Overview
- 1.2 Purpose

## **2. LITERATURE SURVEY**

- 2.1 Existing problem
- 2.2 References
- 2.3 Problem Statement Definition

## **3. IDEATION & PROPOSED SOLUTION**

- 3.1 Empathy Map Canvas
- 3.2 Ideation & Brainstorming

## **4. REQUIREMENT ANALYSIS**

- 4.1 Functional requirement
- 4.2 Non-Functional requirements

## **5. PROJECT DESIGN**

- 5.1 Data Flow Diagrams & User Stories
- 5.2 Solution Architecture

## **6. PROJECT PLANNING & SCHEDULING**

- 6.1 Technical Architecture
- 6.2 Sprint Planning & Estimation
- 6.3 Sprint Delivery Schedule

## **7. CODING & SOLUTIONING**

- 7.1 Feature 1
- 7.2 Feature 2
- 7.3 Database Schema (if Applicable)

## **8. PERFORMANCE TESTING**

- 8.1 Performance Metrics

## **9. RESULTS**

- 9.1 Output Screenshots

## **10. ADVANTAGES & DISADVANTAGES**

## **11. CONCLUSION**

## **12. FUTURE SCOPE**

## **13. APPENDIX**

- Source Code
- GitHub & Project Demo Link

# **1. INTRODUCTION**

## **1.1 Project Overview**

The goal of the "Sign Language Recognition using Deep Learning" project is to create an approachable, real-time system for identifying gestures in American Sign Language (ASL). By utilizing TensorFlow and Keras to enable deep learning, the model can decipher a predetermined set of ASL signs, facilitating communication for those who are hard of hearing. The project's salient characteristics encompass resilient gesture identification in diverse settings, a user-friendly interface, and the possibility of ongoing enhancement via user input and supplementary training materials. The ultimate objective is to support the target audience—which includes educators, persons with hearing impairments, and those interested in sign language communication—in creating a more communicative and inclusive workplace.

## **1.2 Purpose**

The purpose of the 'Sign Language Recognition using Deep Learning' project is to create an innovative solution that enables individuals with hearing impairments to communicate effectively through the recognition of American Sign Language (ASL) gestures.

# **2. LITERATURE SURVEY**

## **2.1 Existing problem**

Extensive research has been conducted on the recognition of American Sign Language (ASL) gestures with the goal of improving communication accessibility for those with hearing impairments. Previous research has identified a number of difficulties in this field. Robust recognition across a variety of environmental conditions, including different lighting and backdrops, is a major difficulty that can greatly affect the performance of recognition models. Furthermore, maintaining accuracy in real-time recognition presents a technological challenge. In certain situations, the feasibility of the state-of-the-art solutions is limited since they frequently call for substantial computational resources. This review of the literature dives into the current issues surrounding ASL recognition and provides the framework for a novel approach that tackles these issues.

## **2.2 References**

- [https://www.researchgate.net/publication/262187093\\_Sign\\_language\\_recognition\\_State\\_of\\_the\\_art](https://www.researchgate.net/publication/262187093_Sign_language_recognition_State_of_the_art)
- <https://ieeexplore.ieee.org/document/9432296>
- [https://github.com/Sri-Tulasi/VIT\\_Morning\\_Slot](https://github.com/Sri-Tulasi/VIT_Morning_Slot)
- <https://www.sciencedirect.com/science/article/pii/S2666990021000471>

## 2.3 Problem Statement Definition

Limited accessibility to American Sign Language (ASL) resources and education hinders effective communication for the deaf community. Our project aims to solve these challenges by implementing an ASL recognition system based on convolution neural networks (CNNs) to improve ASL accessibility and for better communication.

## 3. IDEATION & PROPOSED SOLUTION

### 3.1 Empathy Map Canvas

An empathy map is a simple, easy-to-digest visual that captures knowledge about a user's behaviors and attitudes. It is a useful tool to help us better understand our users. Creating an effective solution requires understanding the true problem and the person who is experiencing it.

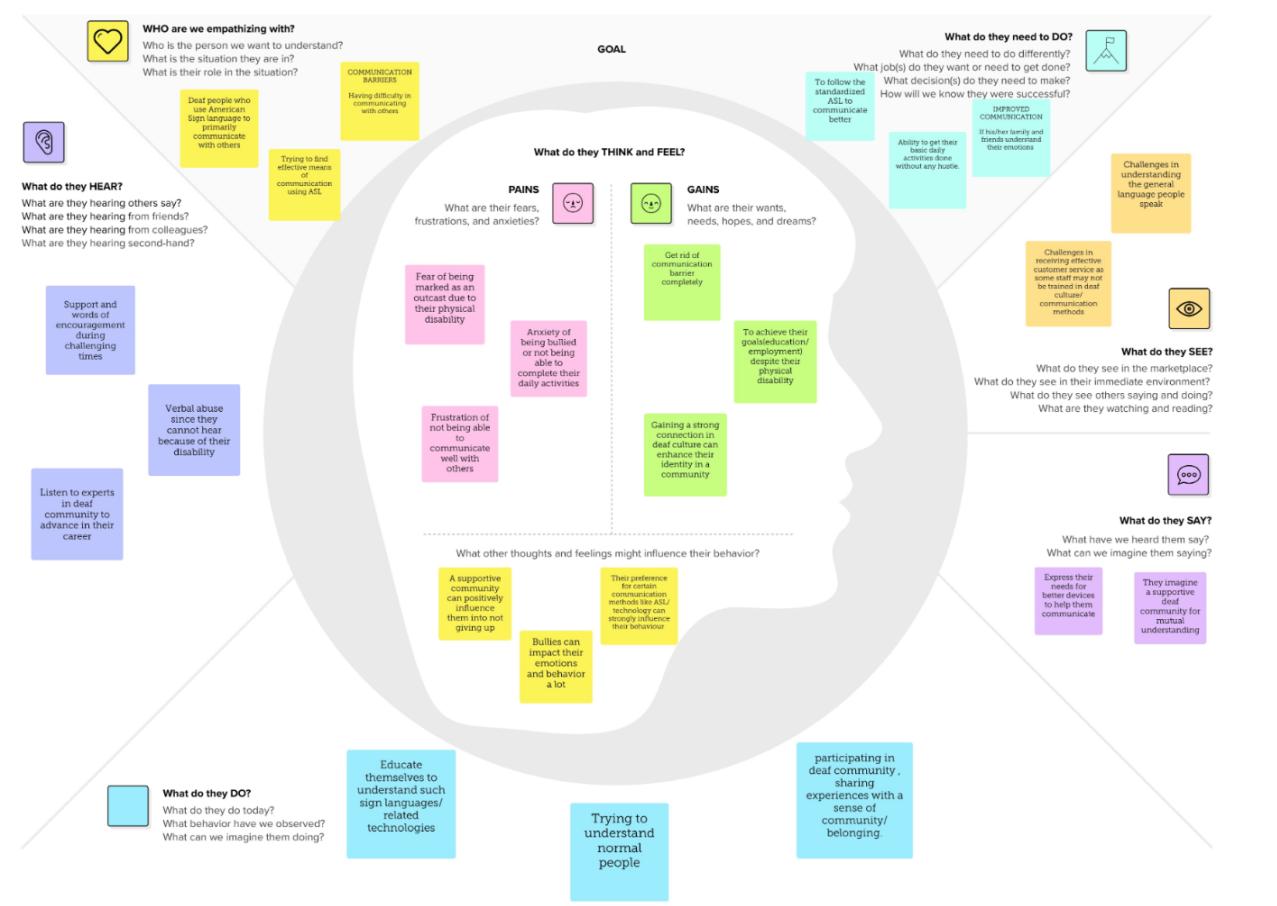
The users of our project would be deaf individuals who use ASL(American Sign Language) to communicate with others. We tried to put ourselves into various situations the deaf might experience and have completed the following empathy canvas map.

#### ASL- Alphabet Image Recognition

ASL(American Sign Language) Alphabet Image Recognition, is an image classification task that aims to recognize the ASL alphabet from images of hand signs. This project involves training a machine learning model to classify images of hand signs corresponding to the 26 letters of the English alphabet, as well as three additional classes for the signs for "space", "delete", and "nothing".

#### TEAM MEMBERS

Pujyam Sathvika  
Ch Rishika Krishna



## 3.2 Ideation & Brainstorming

Brainstorming provides a free and open environment that encourages everyone within a team to participate in the creative thinking process that leads to problem solving.

Prioritizing volume over value, out-of-the-box ideas are welcome and built upon, and all participants are encouraged to collaborate, helping each other develop a rich number of creative solutions.

Our problem statement was “How might we help deaf individuals communicate with the world in a better way?”

The users of our project would be deaf individuals who use ASL(American Sign Language) to communicate with others. We tried to put ourselves into various situations the deaf might experience and tried to come up with various solutions for their problems, grouped the solutions and prioritized them to give us a better understanding of our users.

### Step 1: Team Gathering, Collaboration and Select the Problem Statement

**Brainstorm & idea prioritization**

Use this template in your own brainstorming sessions so your team can unleash their imagination and start shaping concepts even if you're not sitting in the same room.

⌚ 10 minutes to prepare  
⌚ 1 hour to collaborate  
👤 2-8 people recommended

**Before you collaborate**

A little bit of preparation goes a long way with this session. Here's what you need to do to get going.

⌚ 10 minutes

**1 Define your problem statement**

What problem are you trying to solve? Frame your problem as a How Might We statement. This will be the focus of your brainstorm.

⌚ 5 minutes

**PROBLEM**

How might we help deaf individuals communicate with the world in a better way?

**Key rules of brainstorming**

To run a smooth and productive session

- Stay in topic.
- Encourage wild ideas.
- Defer judgment.
- Listen to others.
- Go for volume.
- If possible, be visual.

## Step 2: Brainstorm, Idea Listing and Grouping

2

### Brainstorm

Write down any ideas that come to mind that address your problem statement.

⌚ 10 minutes

3

### Group ideas

**TIP**  
Take turns sharing your ideas while clustering similar or related notes as you go. Once all sticky notes have been grouped, give each cluster a sentence-like label. If a cluster is bigger than six sticky notes, try and break it up into smaller sub-groups.

⌚ 20 minutes

Person 1

- Smart device using microprocessor enabling them to communicate with others
- Hearing aids to help them hear if possible
- Learning lip reading to understand what the other person is saying
- A mobile application which can help them convert text to sign language and vice versa

Person 2

- Creating an online platform with ASL courses to make learning more accessible
- Access to quality healthcare may be limited to deaf people
- some virtual ASL communities to facilitate and support for deaf individuals
- ASL users/deaf people may experience employment related challenges(limited job opportunities)
- Launching some e-books for better knowledge on ASL
- To collaborate with some educational institutions to improve accessibility/ services

**TIP**  
You can select a sticky note and hit the pencil [switch to sketch] icon to start drawing!

### VIRTUAL SERVICES

- Smart device using microprocessor enabling them to communicate with others
- A mobile app which can help them to convert text to sign language and vice versa
- some virtual ASL communities to facilitate and support for deaf individuals
- Creating an online platform with ASL courses to make learning more accessible

### IMPROVEMENTS

- Learning lip reading to understand what the other person is saying
- Hearing aids to help them hear if possible

### AWARENESS/EDUCATION

- Launching some e-books for better knowledge on ASL
- Launching some awareness campaigns/ events to educate people about deaf culture
- To collaborate with some educational institutions to improve accessibility/ services

### CHALLENGES

- Access to quality healthcare may be limited to deaf people
- ASL users/deaf people may experience employment related challenges(limited job opportunities)

**TIP**  
Add customizable tags to sticky notes to make it easier to find, browse, organize, and categorize important ideas as themes within your mind map.

## Step 3: Idea Prioritization

4

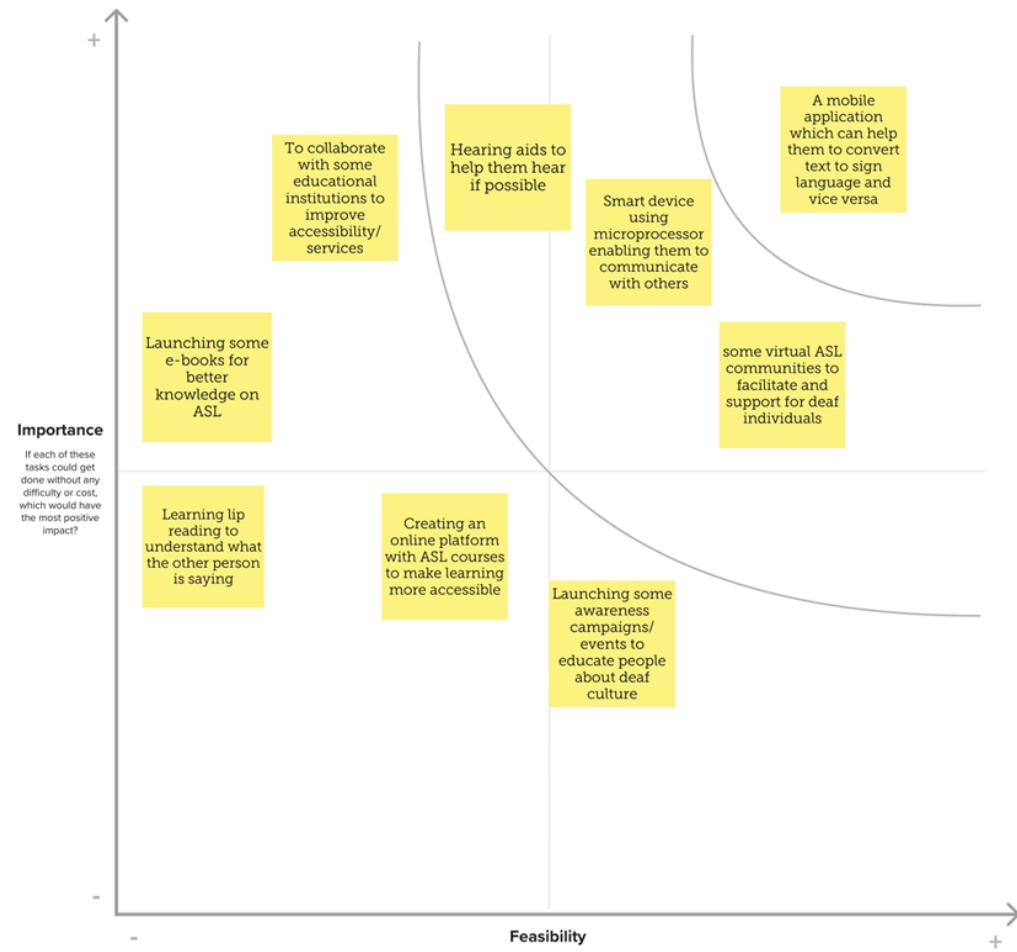
### Prioritize

Your team should all be on the same page about what's important moving forward. Place your ideas on this grid to determine which ideas are important and which are feasible.

⌚ 20 minutes

**TIP**

Participants can use their cursors to point at where sticky notes should go on the grid. The facilitator can confirm the spot by using the laser pointer holding the **H key** on the keyboard.



## **4. REQUIREMENT ANALYSIS**

### **4.1 Functional requirement**

#### **ASL Recognition:**

The system should accurately recognize and interpret American Sign Language gestures. It should be able to recognize a predefined set of ASL signs and translate them into corresponding text or spoken language.

#### **User Interface:**

The system should have a user-friendly interface that enables easy interaction, especially for users with limited technical knowledge.

The interface should include options for users to initiate, pause, or stop the recognition process.

#### **Real-time Processing:**

The system should be capable of processing ASL gestures in real-time, ensuring minimal delay between the user's sign and the system's response.

### **4.2 Non-Functional requirements**

#### **Integration with Communication Tools:**

The system should be designed to integrate seamlessly with existing communication tools and platforms, such as video conferencing applications or messaging apps.

#### **Multi-Gesture Recognition:**

It should support the recognition of multi-gesture sequences to capture complex expressions and sentences in ASL.

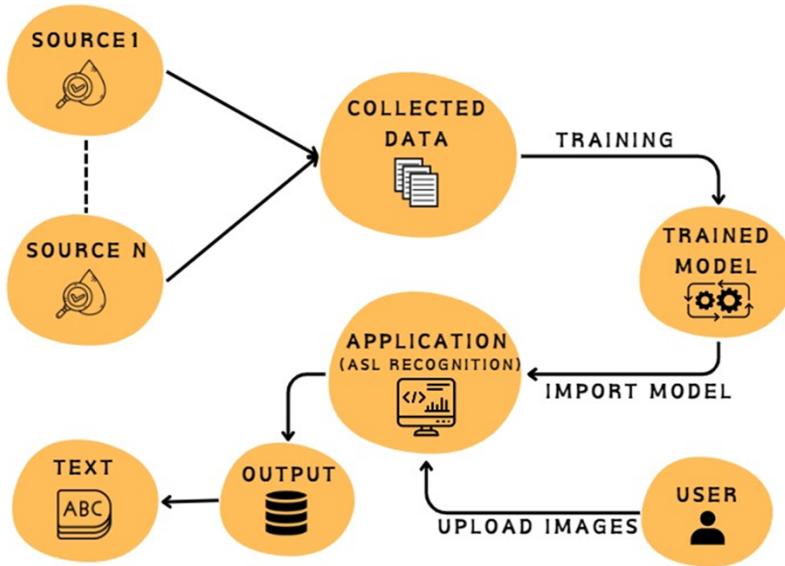
#### **Accessibility:**

Ensure that the user interface is accessible to individuals with varying degrees of physical abilities, including those who may use assistive technologies.

## 5. PROJECT DESIGN

### 5.1 Data Flow Diagrams & User Stories

A Data Flow Diagram (DFD) is a traditional visual representation of the information flows within a system. A neat and clear DFD can depict the right amount of the system requirement graphically. It shows how data enters and leaves the system, what changes the information, and where data is stored.



#### USER STORIES:

User Type	Functional Requirement(Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
As a developer	Projet setup and infrastructure	USN-1	Set up the development environment with the required tools and frameworks to start the ASL project .	Successfully configured with all necessary tools and frameworks.	High	Sprint-1
As a developer	Data collection	USN-2	Gather a diverse and large dataset of images containing all the different classes of alphabets(A, B, C, D, ..... , Z), space, nothing, delete.	Gathered a diverse dataset of images (alphabets).	High	Sprint-1

As a developer	Data Preprocessing	USN-3	Preprocess the collected images by resizing and normalizing. Split the data into training and testing sets.	Preprocessed the dataset.	High	Sprint-2
As a developer	Model Development	USN - 4	Explore different deep learning architectures (CNN) to select the most suitable one for our ASL project.	Explore different DL models.	High	Sprint-2
As a customer	Training	USN-5	As a customer, I want a well trained model and very accurate predictions for the images I upload. Task: Implement data augmentation techniques to improve model's accuracy.	Test the model after training to check accuracy.	Medium	Sprint - 2
As a developer	Integrate the ASL recognition system into applications and services.	USN - 6	As a developer, I want the ASL recognition system to offer an API or web browser for easy accessibility.	Develop a website using HTML, CSS, JS.	Medium	Sprint - 3
Customer	Develop a web-based user interface for the ASL recognition system that provides users with a simple way to interact with the system.	USN - 7	As a customer, I want to access the ASL recognition system through a user-friendly web interface, so I can easily interact with the system without the need to install additional software	Users can access the ASL recognition system through a web browser.	High	Sprint - 3

## 5.2 Solution Architecture

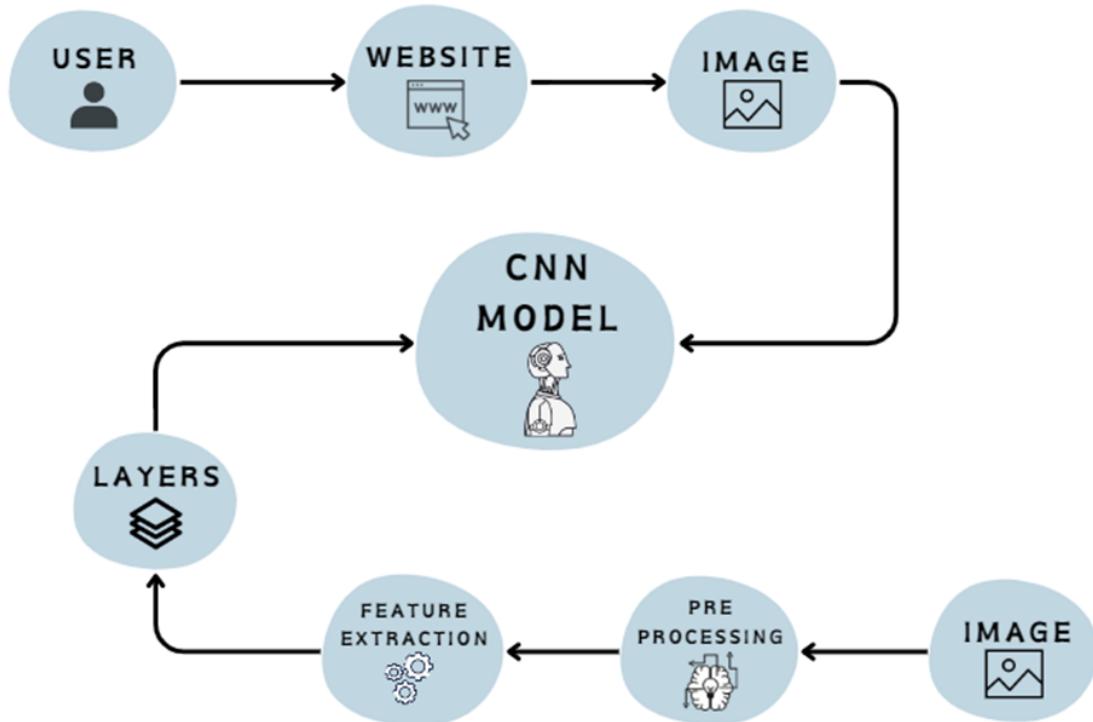
Our problem statement is - "How might we help deaf individuals communicate with the world in a better way?"

Our solution architecture uses machine learning to provide image-based classification for the conversion of sign language to text. We are using CNN(Convolutional Neural Networks) for multi class image classification. Our model can accurately predict the 26

letters of the English alphabet, as well as three additional classes for the signs for "space", "delete", and "nothing".

The steps in our solution architecture include:

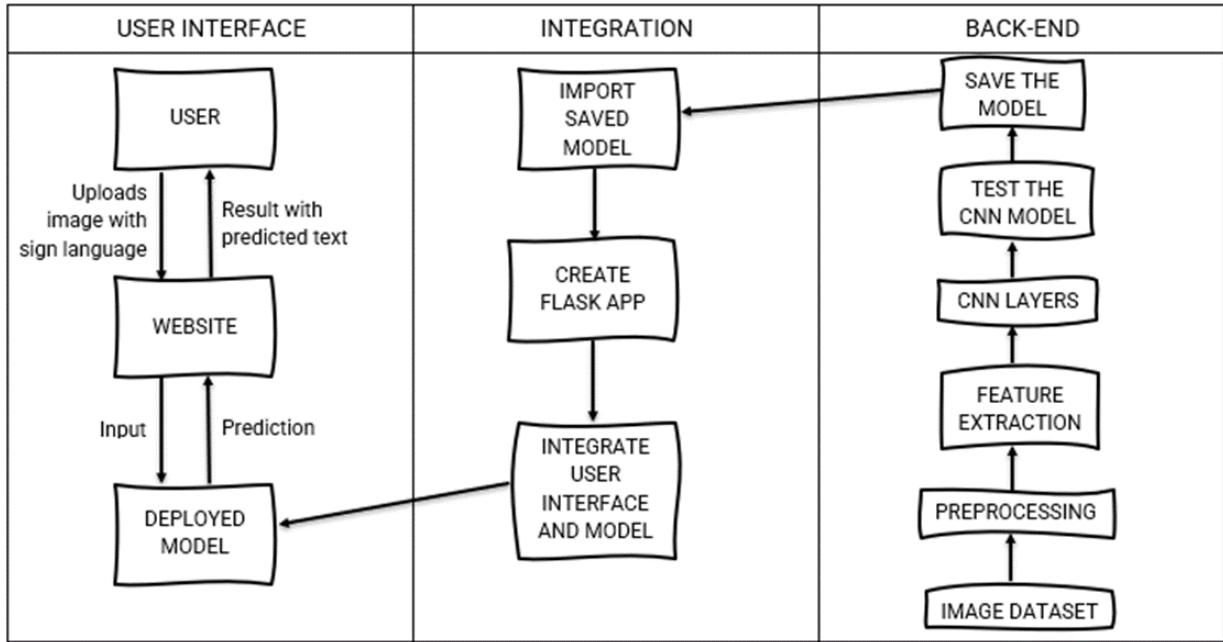
- Data gathering
- Image preprocessing
- Model building
  - Training the model
  - Testing the model
- Sign language prediction
- Website building
- Integrate sign language CNN model using Flask
- Host the website locally
  - Real time analysis



## 6. PROJECT PLANNING & SCHEDULING

### 6.1 Technical Architecture

The technical architecture shows the user interface, the integration, and the back end and how they are tied together to make the project.



### 6.2 Sprint Planning & Estimation

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story points	Priority	Team Members
Sprint-1	Project setup and infrastructure	USN-1	Set up the development environment with the required tools and frameworks to start the ASL project .	1	High	Rishika, Sathvika
Sprint-1	Data collection	USN-2	Gather a diverse and large dataset of images containing all the different classes of alphabets(A, B, C, D, ...., Z), space, nothing, delete.	2	High	Rishika, Sathvika
Sprint-2	Data Preprocessing	USN-3	Preprocess the collected images by resizing and normalizing. Split the data into training and testing sets.	3	High	Sathvika

Sprint-2	Model Development	USN - 4	Explore different deep learning architectures to select the most suitable one(CNN) for our ASL project.	4	High	Rishika
Sprint-2	Training	USN-5	As a customer, I want a well trained model and very accurate predictions for the images I upload. Task: Implement data augmentation techniques to improve model's accuracy.	4	Medium	Sathvika, Rishika
Sprint-3	Integrate the ASL recognition system into applications and services.	USN - 6	As a developer, I want the ASL recognition system to offer an API or web browser for easy accessibility.	6	Medium	Sathvika, Rishika
Sprint-3	Develop a web-based user interface for the ASL recognition system that provides users with a simple way to interact with the system.	USN - 7	As a customer, I want to access the ASL recognition system through a user-friendly web interface, so I can easily interact with the system without the need to install additional software	6	High	Sathvika, Rishika

### 6.3 Sprint Delivery Schedule

Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Point Completed (as on Planned End Date)	Sprint Release Date (Actual)
Sprint-1	3	3 Days	25 Oct 2023	27 Oct 2023	3	27 Oct 2022
Sprint-2	11	5 Days	28 Oct 2023	01 Nov 2023	11	01 Nov 2023
Sprint-3	12	5 Days	02 Nov 2023	06 Nov 2023	12	09 Nov 2023

## 7. CODING & SOLUTIONING (Explain the features added in the project along with code)

### 7.1 Feature 1 - User Interface :

The UI is designed with simplicity and intuitiveness in mind, ensuring that users, regardless of technical expertise, can easily navigate and interact with the system. The interface is designed to be simple and easy to navigate, ensuring a hassle-free experience for users.



### 7.2 Feature 2 - Gesture Recognition :

The system should be able to recognize a diverse set of ASL gestures, including individual signs and multi-gesture sequences. In our project we used the Densenet121 pretrained model for better results.

```
Epoch 34/45
100/100 [=====] - 449s 4s/step - loss: 0.2085 - accuracy: 0.9438
Epoch 35/45
100/100 [=====] - 450s 4s/step - loss: 0.2956 - accuracy: 0.9187
Epoch 36/45
100/100 [=====] - 452s 5s/step - loss: 0.2136 - accuracy: 0.9337
Epoch 37/45
100/100 [=====] - 452s 5s/step - loss: 0.1583 - accuracy: 0.9534
Epoch 38/45
100/100 [=====] - 449s 4s/step - loss: 0.1458 - accuracy: 0.9534
Epoch 39/45
100/100 [=====] - 447s 4s/step - loss: 0.2150 - accuracy: 0.9394
Epoch 40/45
100/100 [=====] - 448s 4s/step - loss: 0.1792 - accuracy: 0.94475
Epoch 41/45
100/100 [=====] - 452s 5s/step - loss: 0.1736 - accuracy: 0.9513
Epoch 42/45
100/100 [=====] - 452s 5s/step - loss: 0.1575 - accuracy: 0.9528
Epoch 43/45
100/100 [=====] - 453s 5s/step - loss: 0.2216 - accuracy: 0.9406
Epoch 44/45
100/100 [=====] - 447s 4s/step - loss: 0.2127 - accuracy: 0.9378
Epoch 45/45
100/100 [=====] - 467s 5s/step - loss: 0.1417 - accuracy: 0.9581
keras.src.callbacks.History at 0x7bf973e20880>
```

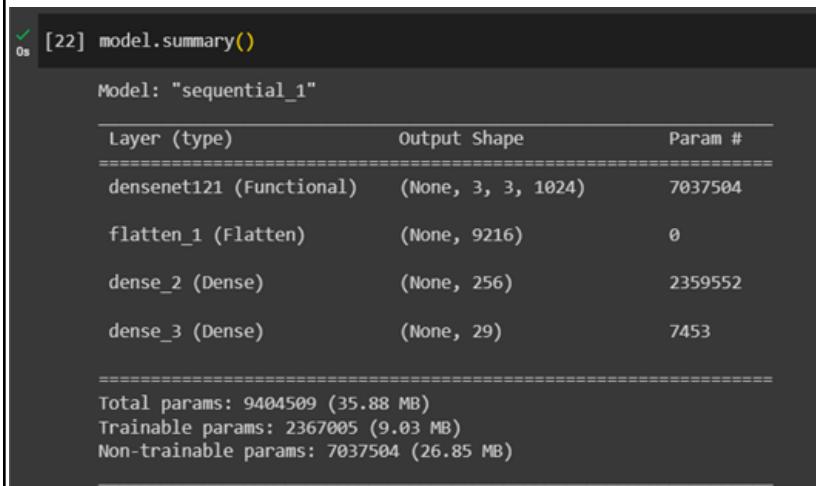
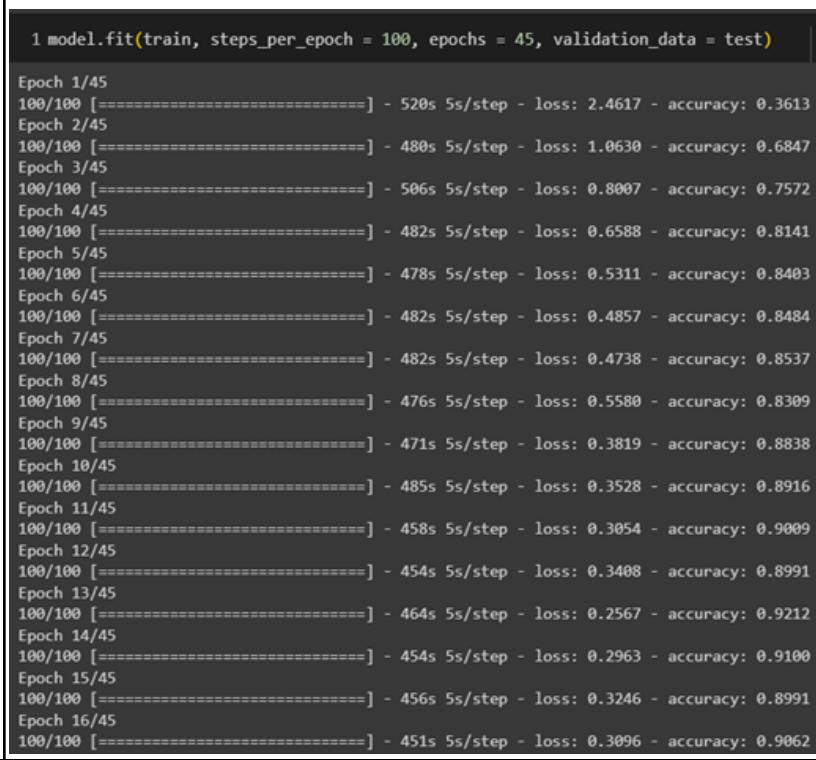
model.save("densenet\_model.h5")

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3079: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This file format is considered legacy and will be removed in a future version.
  saving_api.save_model(
```

Activate Windows

## 8. PERFORMANCE TESTING

### 8.1 Performance Metrics

S.No.	Parameter	Values	Screenshot
1.	Model Summary	Total params: 9404509  Trainable params: 2367005  Non-trainable params: 7037504	 <pre> [22]: model.summary()  Model: "sequential_1" ___ Layer (type)                 Output Shape              Param # denseNet121 (Functional)    (None, 3, 3, 1024)       7037504 flatten_1 (Flatten)          (None, 9216)             0 dense_2 (Dense)              (None, 256)              2359552 dense_3 (Dense)              (None, 29)               7453 ___ Total params: 9404509 (35.88 MB) Trainable params: 2367005 (9.03 MB) Non-trainable params: 7037504 (26.85 MB) </pre>
2.	Accuracy	Final training accuracy – 0.9581	 <pre> 1 model.fit(train, steps_per_epoch = 100, epochs = 45, validation_data = test)  Epoch 1/45 100/100 [=====] - 520s 5s/step - loss: 2.4617 - accuracy: 0.3613 Epoch 2/45 100/100 [=====] - 480s 5s/step - loss: 1.0630 - accuracy: 0.6847 Epoch 3/45 100/100 [=====] - 506s 5s/step - loss: 0.8007 - accuracy: 0.7572 Epoch 4/45 100/100 [=====] - 482s 5s/step - loss: 0.6588 - accuracy: 0.8141 Epoch 5/45 100/100 [=====] - 478s 5s/step - loss: 0.5311 - accuracy: 0.8403 Epoch 6/45 100/100 [=====] - 482s 5s/step - loss: 0.4857 - accuracy: 0.8484 Epoch 7/45 100/100 [=====] - 482s 5s/step - loss: 0.4738 - accuracy: 0.8537 Epoch 8/45 100/100 [=====] - 476s 5s/step - loss: 0.5580 - accuracy: 0.8309 Epoch 9/45 100/100 [=====] - 471s 5s/step - loss: 0.3819 - accuracy: 0.8838 Epoch 10/45 100/100 [=====] - 485s 5s/step - loss: 0.3528 - accuracy: 0.8916 Epoch 11/45 100/100 [=====] - 458s 5s/step - loss: 0.3054 - accuracy: 0.9009 Epoch 12/45 100/100 [=====] - 454s 5s/step - loss: 0.3408 - accuracy: 0.8991 Epoch 13/45 100/100 [=====] - 464s 5s/step - loss: 0.2567 - accuracy: 0.9212 Epoch 14/45 100/100 [=====] - 454s 5s/step - loss: 0.2963 - accuracy: 0.9100 Epoch 15/45 100/100 [=====] - 456s 5s/step - loss: 0.3246 - accuracy: 0.8991 Epoch 16/45 100/100 [=====] - 451s 5s/step - loss: 0.3096 - accuracy: 0.9062 </pre>

## Model Summary:

```
✓ [22] model.summary()  
  
Model: "sequential_1"  


| Layer (type)                             | Output Shape       | Param # |
|------------------------------------------|--------------------|---------|
| densenet121 (Functional)                 | (None, 3, 3, 1024) | 7037504 |
| flatten_1 (Flatten)                      | (None, 9216)       | 0       |
| dense_2 (Dense)                          | (None, 256)        | 2359552 |
| dense_3 (Dense)                          | (None, 29)         | 7453    |
| <hr/>                                    |                    |         |
| Total params: 9404509 (35.88 MB)         |                    |         |
| Trainable params: 2367005 (9.03 MB)      |                    |         |
| Non-trainable params: 7037504 (26.85 MB) |                    |         |


```

## Training accuracy:

```
1 model.fit(train, steps_per_epoch = 100, epochs = 45, validation_data = test)  
  
Epoch 1/45  
100/100 [=====] - 520s 5s/step - loss: 2.4617 - accuracy: 0.3613  
Epoch 2/45  
100/100 [=====] - 480s 5s/step - loss: 1.0630 - accuracy: 0.6847  
Epoch 3/45  
100/100 [=====] - 506s 5s/step - loss: 0.8007 - accuracy: 0.7572  
Epoch 4/45  
100/100 [=====] - 482s 5s/step - loss: 0.6588 - accuracy: 0.8141  
Epoch 5/45  
100/100 [=====] - 478s 5s/step - loss: 0.5311 - accuracy: 0.8403  
Epoch 6/45  
100/100 [=====] - 482s 5s/step - loss: 0.4857 - accuracy: 0.8484  
Epoch 7/45  
100/100 [=====] - 482s 5s/step - loss: 0.4738 - accuracy: 0.8537  
Epoch 8/45  
100/100 [=====] - 476s 5s/step - loss: 0.5580 - accuracy: 0.8309  
Epoch 9/45  
100/100 [=====] - 471s 5s/step - loss: 0.3819 - accuracy: 0.8838  
Epoch 10/45  
100/100 [=====] - 485s 5s/step - loss: 0.3528 - accuracy: 0.8916  
Epoch 11/45  
100/100 [=====] - 458s 5s/step - loss: 0.3054 - accuracy: 0.9009  
Epoch 12/45  
100/100 [=====] - 454s 5s/step - loss: 0.3408 - accuracy: 0.8991  
Epoch 13/45  
100/100 [=====] - 464s 5s/step - loss: 0.2567 - accuracy: 0.9212  
Epoch 14/45  
100/100 [=====] - 454s 5s/step - loss: 0.2963 - accuracy: 0.9100  
Epoch 15/45  
100/100 [=====] - 456s 5s/step - loss: 0.3246 - accuracy: 0.8991  
Epoch 16/45  
100/100 [=====] - 451s 5s/step - loss: 0.3096 - accuracy: 0.9062
```

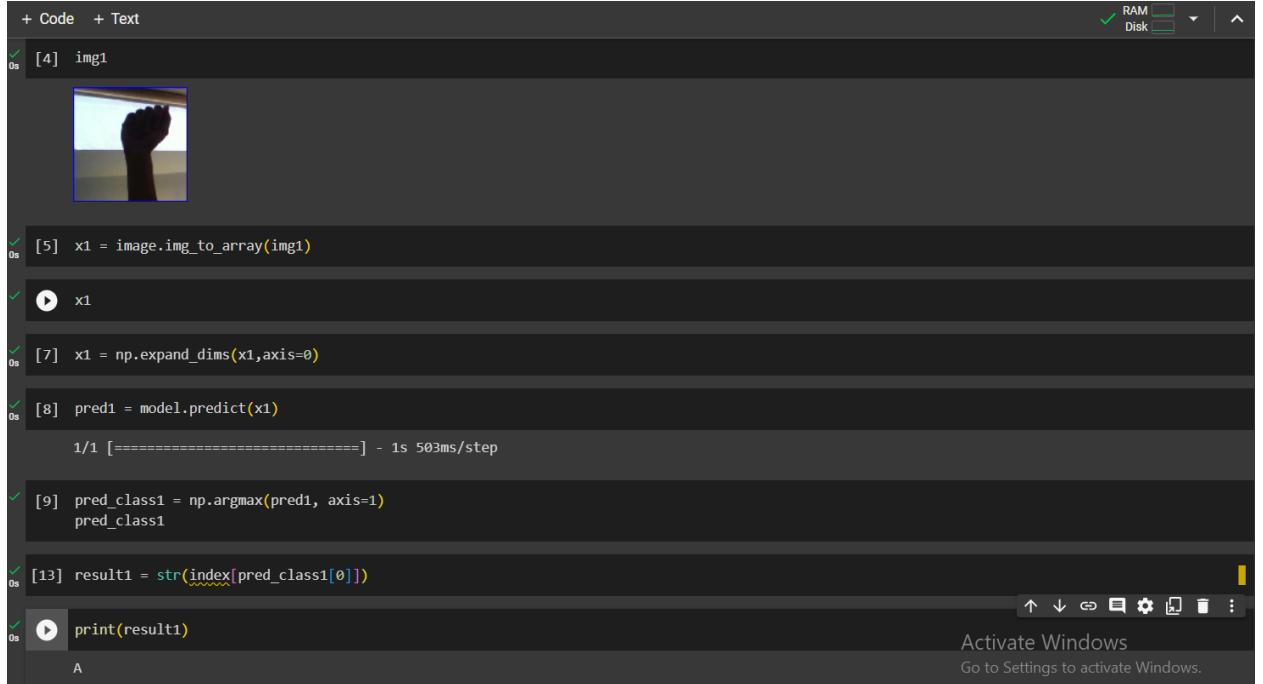
```
model.fit(train, steps_per_epoch = 100, epochs = 45, validation_data = test)

Epoch 16/45
100/100 [=====] - 451s 5s/step - loss: 0.3096 - accuracy: 0.9062
Epoch 17/45
100/100 [=====] - 460s 5s/step - loss: 0.2730 - accuracy: 0.9216
Epoch 18/45
100/100 [=====] - 463s 5s/step - loss: 0.2688 - accuracy: 0.9228
Epoch 19/45
100/100 [=====] - 463s 5s/step - loss: 0.2482 - accuracy: 0.9222
Epoch 20/45
100/100 [=====] - 469s 5s/step - loss: 0.3183 - accuracy: 0.8988
Epoch 21/45
100/100 [=====] - 475s 5s/step - loss: 0.2512 - accuracy: 0.9237
Epoch 22/45
100/100 [=====] - 455s 5s/step - loss: 0.2507 - accuracy: 0.9303
Epoch 23/45
100/100 [=====] - 460s 5s/step - loss: 0.2350 - accuracy: 0.9319
Epoch 24/45
100/100 [=====] - 458s 5s/step - loss: 0.2460 - accuracy: 0.9231
Epoch 25/45
100/100 [=====] - 469s 5s/step - loss: 0.1844 - accuracy: 0.9375
Epoch 26/45
100/100 [=====] - 460s 5s/step - loss: 0.2047 - accuracy: 0.9319
Epoch 27/45
100/100 [=====] - 461s 5s/step - loss: 0.2295 - accuracy: 0.9316
Epoch 28/45
100/100 [=====] - 466s 5s/step - loss: 0.2013 - accuracy: 0.9378
Epoch 29/45
100/100 [=====] - 460s 5s/step - loss: 0.2113 - accuracy: 0.9362
Epoch 30/45
100/100 [=====] - 457s 5s/step - loss: 0.2060 - accuracy: 0.9428
Epoch 31/45
100/100 [=====] - 449s 4s/step - loss: 0.2795 - accuracy: 0.9234
Epoch 32/45
```

```
Epoch 32/45
100/100 [=====] - 454s 5s/step - loss: 0.1805 - accuracy: 0.9463
Epoch 33/45
100/100 [=====] - 448s 4s/step - loss: 0.1723 - accuracy: 0.9466
Epoch 34/45
100/100 [=====] - 449s 4s/step - loss: 0.2085 - accuracy: 0.9438
Epoch 35/45
100/100 [=====] - 450s 4s/step - loss: 0.2956 - accuracy: 0.9187
Epoch 36/45
100/100 [=====] - 452s 5s/step - loss: 0.2136 - accuracy: 0.9337
Epoch 37/45
100/100 [=====] - 452s 5s/step - loss: 0.1503 - accuracy: 0.9534
Epoch 38/45
100/100 [=====] - 449s 4s/step - loss: 0.1458 - accuracy: 0.9534
Epoch 39/45
100/100 [=====] - 447s 4s/step - loss: 0.2150 - accuracy: 0.9394
Epoch 40/45
100/100 [=====] - 448s 4s/step - loss: 0.1792 - accuracy: 0.9475
Epoch 41/45
100/100 [=====] - 452s 5s/step - loss: 0.1736 - accuracy: 0.9513
Epoch 42/45
100/100 [=====] - 452s 5s/step - loss: 0.1575 - accuracy: 0.9528
Epoch 43/45
100/100 [=====] - 453s 5s/step - loss: 0.2216 - accuracy: 0.9406
Epoch 44/45
100/100 [=====] - 447s 4s/step - loss: 0.2127 - accuracy: 0.9378
Epoch 45/45
100/100 [=====] - 467s 5s/step - loss: 0.1417 - accuracy: 0.9581
<keras.src.callbacks.History at 0x7bf973e20880>
```

## 9. RESULTS

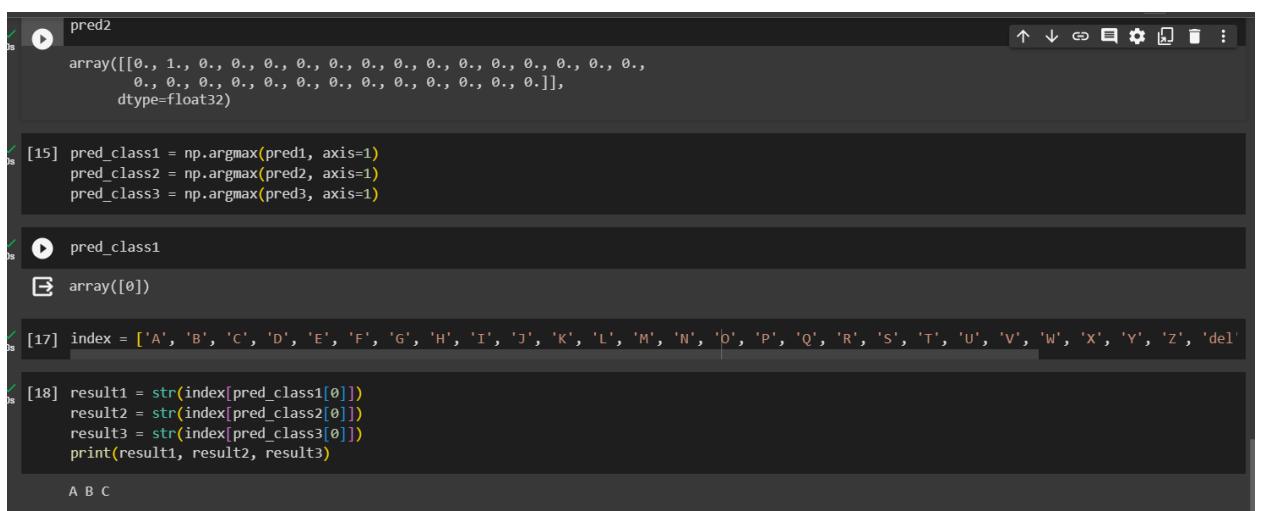
### 9.1 Output Screenshots



The screenshot shows a Jupyter Notebook interface with two code cells. The first cell displays an image of a hand. The second cell contains Python code for image processing and model prediction.

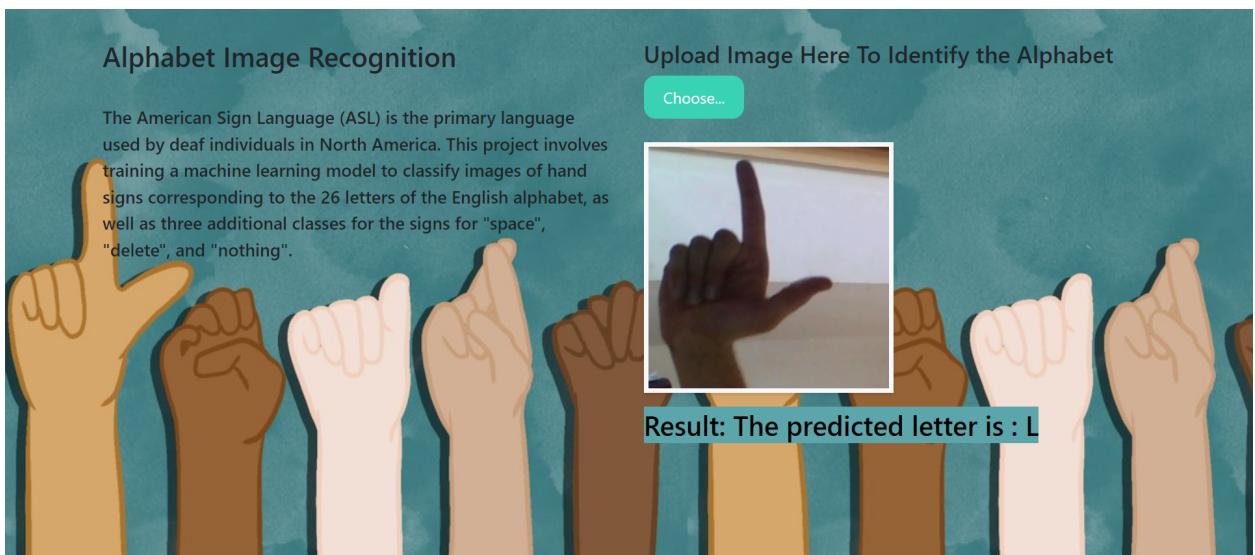
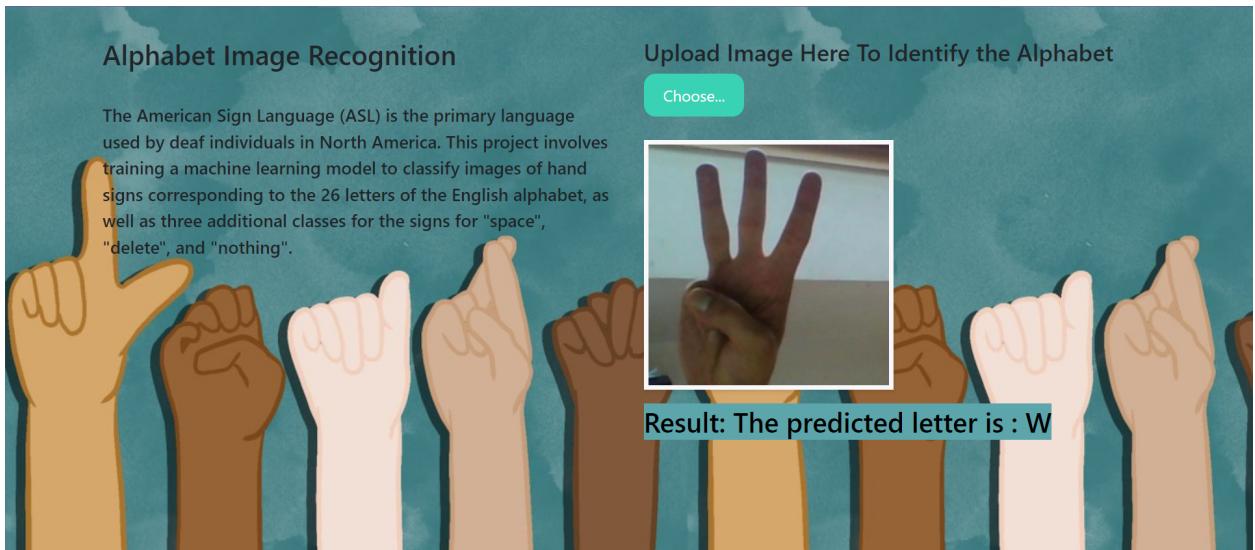
```
+ Code + Text  
[4] img1  
  
[5] x1 = image.img_to_array(img1)  
[6] x1  
[7] x1 = np.expand_dims(x1, axis=0)  
[8] pred1 = model.predict(x1)  
1/1 [=====] - 1s 503ms/step  
[9] pred_class1 = np.argmax(pred1, axis=1)  
pred_class1  
[10] result1 = str(index[pred_class1[0]])  
[11] print(result1)  
A
```

Activate Windows  
Go to Settings to activate Windows.

The screenshot shows a Jupyter Notebook interface with two code cells. The first cell displays an array of three images. The second cell contains Python code for processing multiple images and extracting specific characters.

```
[12] pred2  
array([[0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,  
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]],  
      dtype=float32)  
[13] pred_class1 = np.argmax(pred1, axis=1)  
pred_class2 = np.argmax(pred2, axis=1)  
pred_class3 = np.argmax(pred3, axis=1)  
[14] pred_class1  
array([0])  
[15] index = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z', 'del']  
[16] result1 = str(index[pred_class1[0]])  
result2 = str(index[pred_class2[0]])  
result3 = str(index[pred_class3[0]])  
print(result1, result2, result3)  
A B C
```



## 10. ADVANTAGES & DISADVANTAGES

### Advantages:

1. By giving people with hearing impairments a dependable tool for deciphering American Sign Language (ASL) motions, the project greatly improves accessibility and promotes improved communication.
2. Real-time ASL gesture recognition is incorporated to provide quick and flexible interpretation, making the user experience more responsive and smooth.
3. Because of its architecture, the model can be continuously improved over time by adding more training data and incorporating user feedback, which guarantees its accuracy and adaptability.

## **Disadvantages:**

1. The deep learning model's requirement for real-time processing in environments with limited resources could present difficulties, as it requires a significant amount of computational power to achieve optimal performance.
2. The vocabulary that can be used for communication is limited by the current implementation, which concentrates on a predetermined set of ASL gestures. More training and data would be needed to expand the vocabulary.
3. Deep learning models with high levels of complexity, like VGG16/Densenet121, may require more processing power during training and inference, and their length may cause these issues.
4. The model's performance can be influenced by certain environmental conditions, such as lighting and background variations affecting accuracy in some cases.

## **11. CONCLUSION**

A major step towards promoting inclusive communication for people with hearing impairments has been made with the completion of the "Sign Language Recognition using Deep Learning" project. Real-time American Sign Language (ASL) gesture recognition is implemented in this project to fulfill the urgent need for accessible tools that close communication gaps.

With the conclusion of the "Sign Language Recognition using Deep Learning" project, a significant step has been made towards encouraging inclusive communication for those with hearing impairments. This project implements real-time gesture recognition in American Sign Language (ASL) to meet the pressing need for accessible tools that bridge communication gaps.

## **12. FUTURE SCOPE**

### **Vocabulary Expansion:**

In order to create a more complete and adaptable communication tool, future versions of the project may concentrate on increasing the vocabulary of ASL gestures that are recognised.

### **Adaptive Learning:**

By enabling the model to continuously change and get better based on user interactions and feedback, adaptive learning mechanisms could improve the model's performance.

## 13. APPENDIX

### Source Code - Model building

```
!pip install -q kaggle
!mkdir ~/.kaggle
!cp kaggle.json ~/.kaggle
!kaggle datasets download -d
debashishsau/aslamerican-sign-language-aplhabet-dataset
!unzip /content/aslamerican-sign-language-aplhabet-dataset.zip

from tensorflow.keras.preprocessing.image import ImageDataGenerator
trainpath = "/content/ASL_Alphabet_Dataset/asl_alphabet_train"
testpath = "/content/ASL_Alphabet_Dataset/asl_alphabet_test"
train_datagen = ImageDataGenerator(rescale = 1. / 255, zoom_range = 0.2,
shear_range = 0.2)
test_datagen = ImageDataGenerator(rescale = 1. / 255)
train = train_datagen.flow_from_directory(trainpath, target_size = (108, 108), batch_size = 32)
test = test_datagen.flow_from_directory(testpath, target_size = (108, 108), batch_size = 32)
train.class_indices
from keras.applications import DenseNet121
from keras.models import Sequential
from keras.layers import Flatten, Dense
from keras.optimizers import Adam

densenet_model = DenseNet121(weights='imagenet', include_top=False,
input_shape=(108, 108, 3))
model = Sequential()
model.add(densenet_model)
densenet_model.trainable = False
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dense(29, activation='softmax'))
model.summary()
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
model.fit(train, steps_per_epoch = 100, epochs = 45, validation_data = test)
model.save("densenet_model.h5")
```

## Source Code - Model testing

```
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
import numpy as np
model = load_model("/content/densenet_model.h5")
img2 = image.load_img(r"/content/B_test.jpg", target_size = (108, 108))
x2 = image.img_to_array(img2)
x2 = np.expand_dims(x2, axis=0)
pred2 = model.predict(x2)
pred_class2 = np.argmax(pred2, axis=1)
pred_class2
print(result2)
```

## GitHub & Project Demo Link

Github -

<https://github.com/smartinternz02/SI-GuidedProject-613088-1698898208>

Project Demo -

<https://drive.google.com/file/d/17U5NzrywPw3rNTNeUHwq0ZMYqC7e4AfK/view?usp=sharing>