American Sign Language Alphabet Image Recognition

Project Done By:-

Ananth Anand PB (21BCE1576)

Adwyth Sumesh (21BCE5604)

Joshua Jose (21BCE5142)

Table of Contents

1. Introduction---------- 3
1.1 Project Overview---------- 3
1.2 Purpose---------- 4
2. Literature Survey---------- 5
2.1 Existing problem---------- 5
2.2 References---------- 5
2.3 Problem Statement Definition---------- 5
3. Ideation and Proposed Solution----------6
3.1 Empathy Map Canvas----------6
3.2 Ideation & Brainstorming---------- 8
4. Requirement Analysis---------- 9
4.1 Functional requirements ---------- 9
4.2 Non-Functional Requirements---------- 9
5. Project Design---------- 10
5.1 Data Flow Diagrams & User Stories---------- 10
5.2 Solution Architecture---------- 13
6. Project Planning and Scheduling---------- 14
6.1 Technical Architecture---------- 14
6.2 Sprint Planning & Estimation---------- 14
6.3 Sprint Delivery Schedule----------15
7. Coding and Solutions---------- 16
7.1 Feature 1: Real-time text formation:---------- 16
7.2 Feature 2: Text-to-Speech conversion:----------18
8. Performance Testing---------- 18
9. Results---------- 19
10. Advantages and Disadvantages---------- 20
10.1 Advantages:---------- 20
10.2 Disadvantages:---------- 21
11. Conclusion---------- 22
12. Future Scope---------- 22
13. Appendix:---------- 23
13.1 Source Code:---------- 23
Model Building Code:---------- 23
Testing:---------- 34
WSGI Flask Server Code:---------- 36
Classes.txt File:---------- 38
Home-Page HTML file:---------- 38
Home-Page CSS file:---------- 39
Demo-Page HTML file:---------- 40
Demo-Page CSS file:---------- 42
13.2 Git Hub Link:---------- 43

1. Introduction

1.1 Project Overview

ASL Alphabet Image Recognition is a project aimed at developing a machine learning model capable of recognizing the American Sign Language (ASL) alphabet from images of hand symbols.

The project was motivated by the need to improve communication between the deaf and hard-of-hearing communities, with ASL being the primary language used by deaf people in North America.

- The project includes the following steps: Data collection: The project uses a dataset of 87,000 images of hand symbols corresponding to the 26 letters of the English alphabet, as well as three additional classes supplement the symbols "space", “delete”, and nothing”.
- The dataset is publicly available on Kaggle.
- Data Preprocessing: The project applies various image processing techniques to a dataset, such as resizing, cropping, grayscale conversion, histogram equalization, and noise reduction, to improve image quality and consistency.
- Model training: Project to train a convolutional neural network (CNN) model to classify hand symbol images.
- The CNN model consists of multiple convolutional, pooling, subtraction, and fully connected layers.
- The project uses TensorFlow as a framework to build and train models.
- Model Evaluation: The project evaluates the performance of a model trained on a test set of unseen images.
- The project uses various metrics, such as accuracy, and precision to measure the effectiveness of the model.
- The project also visualizes the confusion matrix and the classification report to analyze the model's strengths and weaknesses.
- Model deployment: The project deploys the trained model as a web application that can recognize the ASL alphabet from real-time video streams.
- The web application uses Flask as the backend framework, HTML, CSS, JavaScript, and Local File System as the file storage.

The web application allows users to capture video from their webcam and displays the predicted letter on the screen.

1.2 Purpose

The goal of the ASL Alphabet Image Recognition project is to create a technology solution that enables seamless communication between deaf people who use American Sign Language (ASL) and people who do not understand the language.

Leveraging the capabilities of a machine learning model trained on a comprehensive dataset of ASL hand signs, the project seeks to accurately classify images of the ASL alphabet and translate them into text.

This translation facilitates understanding and interaction, thereby reducing communication barriers that often isolate the deaf community.

The project also aims to provide a real-time application that can be used in a variety of contexts, such as educational environments, public services, and personal interactions, where translation Quick and accurate ASL will be beneficial.

By integrating this technology into a user-friendly interface, the project strives to promote inclusivity and equal access to information for all individuals, regardless of their hearing ability.

Additionally, the project is an educational tool for people interested in learning ASL, providing a convenient way to practice and improve their sign language skills through interaction with the app.

use.

Overall, the ASL Alphabet Image Recognition project represents a step forward in assistive technology, with the potential to significantly impact the lives of many people.

2. Literature Survey

2.1 Existing problem

The existing problem that the ASL Alphabet Image Recognition project addresses is the lack of effective communication between deaf individuals who use American Sign Language (ASL) and those who do not understand it. ASL is a visual language that combines hand gestures, facial expressions, and body movements to convey meaning. However, not many people are familiar with ASL, which creates a communication gap between the deaf and hearing communities.

This communication gap can lead to various challenges and disadvantages for deaf individuals, such as social isolation, limited access to education, employment, and public services, and reduced quality of life. Moreover, learning ASL can be difficult and time-consuming for those who want to communicate with deaf individuals, as it requires a lot of practice and feedback. Therefore, there is a need for a technological solution that can facilitate communication between deaf individuals who use ASL and those who do not understand it.

2.2 References

The references for the ASL Alphabet Image Recognition project are:

- Aditya Taparia, ASL Alphabet Recognition, GitHub.
- Grassknoted, ASL Alphabet, Kaggle.
- Max Agar, ASL-Alphabet Image Classification, GitHub.
- S. S. Rautaray and A. Agrawal, American Sign Language Alphabet Recognition using Deep Learning, arXiv.
- Smart Internz - ASL alphabet Image Recognition.docx.pdf - Google Drive

2.3 Problem Statement Definition

Deaf individuals who use American Sign Language (ASL) face communication barriers with those who do not understand it, which can limit their social, educational, and professional opportunities. Existing solutions, such as interpreters, are often costly, scarce, or unavailable. Therefore, there is a need for a low-cost, accessible, and accurate technology that can recognize the ASL alphabet from images of hand signs and translate them into text. This technology would enable seamless communication between deaf

individuals who use ASL and those who do not understand it, as well as provide a convenient way to learn and practice ASL.

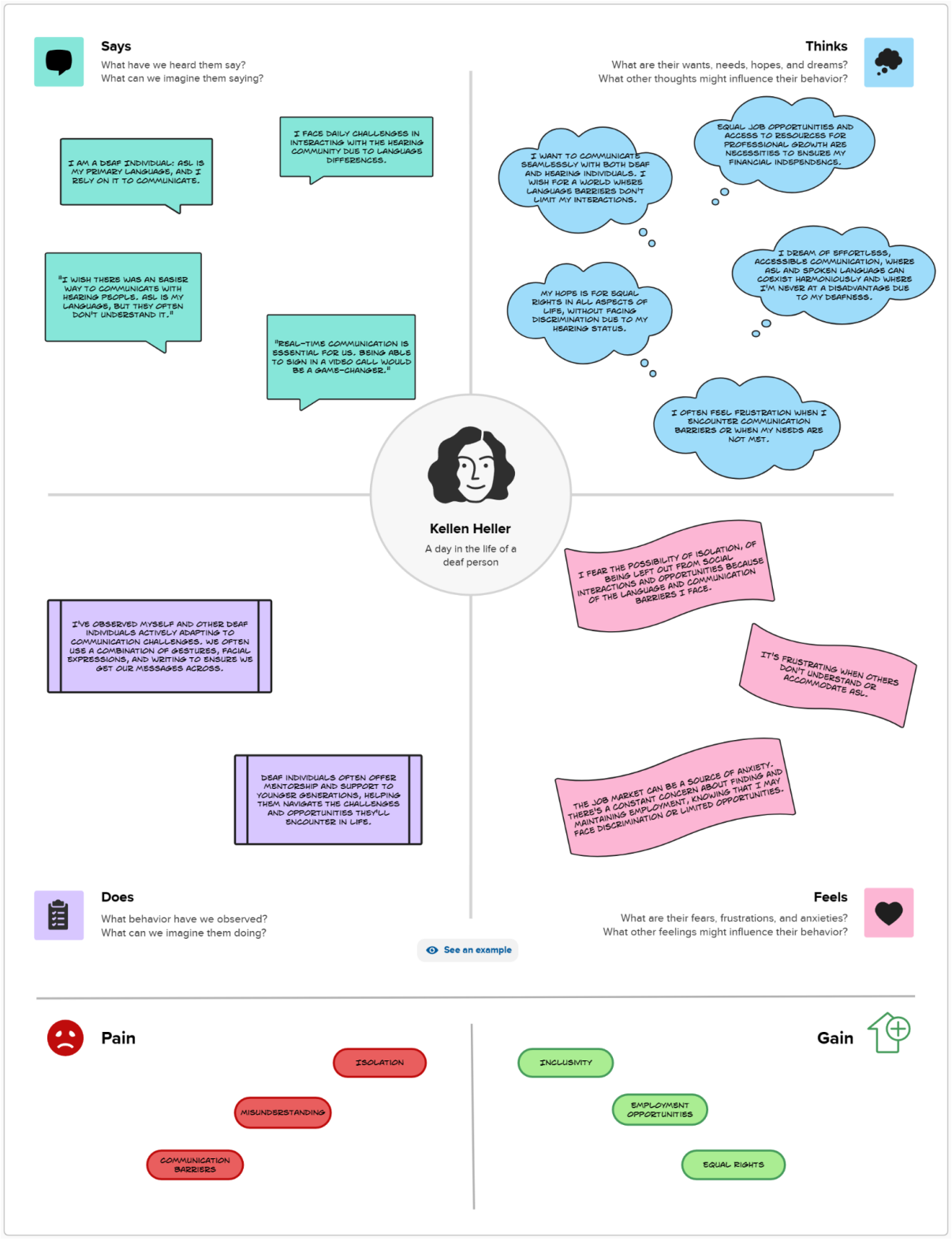
This problem statement definition describes the current situation or problem as the communication gap between the deaf and hearing communities due to the lack of familiarity with ASL. It describes the desired outcome or goal as the development of a technology that can recognize the ASL alphabet from images of hand signs and translate them into text. It describes the gap or difference as the lack of a low-cost, accessible, and accurate solution that can facilitate communication and learning of ASL. It also explains the motivation and significance of the project, as well as the potential benefits and impacts of the technology.

3. Ideation and Proposed Solution

3.1 Empathy Map Canvas

An empathy map is a visualization tool that helps you understand and empathize with your target users or customers. It is a tool often used in design thinking and user-centred design to help teams better understand and empathize with their target audience.

An empathy map typically consists of a square divided into four quadrants, with the user or customer at the centre. Each quadrant represents a different aspect of the user's mindset, such as what they think, feel, say, do, see, or hear. By filling in each quadrant with data collected from user research, interviews, surveys, or observations, you can create a comprehensive picture of the user's thoughts, feelings, needs, and behaviours.

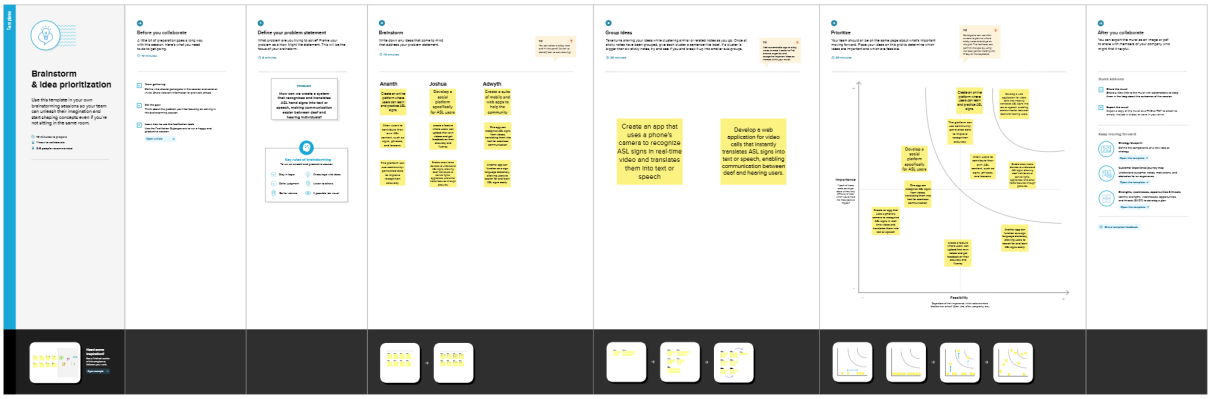

Says
What have we heard them say?
What can we imagine them saying?
I AM A DEAF INDIVIDUAL: ASL IS MY PRIMARY LANGUAGE, AND I RELY ON IT TO COMMUNICATE.
I FACE DAILY CHALLENGES IN INTERACTING WITH THE HEARING COMMUNITY DUE TO LANGUAGE DIFFERENCES.
"I WISH THERE WAS AN EASIER WAY TO COMMUNICATE WITH HEARING PEOPLE. ASL IS MY LANGUAGE, BUT THEY OFTEN DON'T UNDERSTAND IT."
"REAL-TIME COMMUNICATION IS ESSENTIAL FOR US. BEING ABLE TO SIGN IN A VIDEO CALL WOULD BE A GAME-CHANGER."
Thinks
What are their wants, needs, hopes, and dreams?
What other thoughts might influence their behavior?
I WANT TO COMMUNICATE SEAMLESSLY WITH BOTH DEAF AND HEARING INDIVIDUALS. I WISH FOR A WORLD WHERE LANGUAGE BARRIERS DON'T LIMIT MY INTERACTIONS.
EQUAL JOB OPPORTUNITIES AND ACCESS TO RESOURCES FOR PROFESSIONAL GROWTH ARE NECESSITIES TO ENSURE MY FINANCIAL INDEPENDENCE.
I DREAM OF EFFORTLESS, ACCESSIBLE COMMUNICATION, WHERE ASL AND SPOKEN LANGUAGE CAN COEXIST HARMONIOUSLY AND WHERE I'M NEVER AT A DISADVANTAGE DUE TO MY DEAFNESS.
MY HOPE IS FOR EQUAL RIGHTS IN ALL ASPECTS OF LIFE, WITHOUT FACING DISCRIMINATION DUE TO MY HEARING STATUS.
I OFTEN FEEL FRUSTRATION WHEN I ENCOUNTER COMMUNICATION BARRIERS OR WHEN MY NEEDS ARE NOT MET.
Kellen Heller
A day in the life of a deaf person
I FEAR THE POSSIBILITY OF ISOLATION, OF BEING LEFT OUT FROM SOCIAL INTERACTIONS AND OPPORTUNITIES BECAUSE OF THE LANGUAGE AND COMMUNICATION BARRIERS I FACE.
IT'S FRUSTRATING WHEN OTHERS DON'T UNDERSTAND OR ACCOMMODATE ASL.
THE JOB MARKET CAN BE A SOURCE OF ANXIETY. THERE'S A CONSTANT CONCERN ABOUT FINDING AND MAINTAINING EMPLOYMENT, KNOWING THAT I MAY FACE DISCRIMINATION OR LIMITED OPPORTUNITIES.
I'VE OBSERVED MYSELF AND OTHER DEAF INDIVIDUALS ACTIVELY ADAPTING TO COMMUNICATION CHALLENGES. WE OFTEN USE A COMBINATION OF GESTURES, FACIAL EXPRESSIONS, AND WRITING TO ENSURE WE GET OUR MESSAGES ACROSS.
DEAF INDIVIDUALS OFTEN OFFER MENTORSHIP AND SUPPORT TO YOUNGER GENERATIONS, HELPING THEM NAVIGATE THE CHALLENGES AND OPPORTUNITIES THEY'LL ENCOUNTER IN LIFE.
Does
What behavior have we observed?
What can we imagine them doing?
Feels
What are their fears, frustrations, and anxieties?
What other feelings might influence their behavior?
See an example
Pain
ISOLATION
MISUNDERSTANDING
COMMUNICATION BARRIERS
Gain
INCLUSIVITY
EMPLOYMENT OPPORTUNITIES
EQUAL RIGHTS

3.2 Ideation & Brainstorming

Ideation and brainstorming are creative processes that help you generate and explore different ideas and solutions for your project. Ideation and brainstorming can be done individually or in groups, using various techniques and methods. Some examples of ideation and brainstorming techniques are:

- Braindump: This technique involves writing down or sketching all the ideas that come to your mind, without filtering or judging them. You can use a paper, a whiteboard, a sticky note, or a digital tool to capture your ideas. The goal is to generate as many ideas as possible in a short time, and then review and refine them later.
- Mind map: This technique involves creating a visual representation of your ideas and how they are related to each other. You can start with a central topic or question, and then branch out to subtopics or subquestions, and so on. You can use colours, symbols, images, or words to represent your ideas. The goal is to organize and structure your ideas logically and intuitively.
- SCAMPER: This technique involves using a set of questions to modify or improve an existing idea or solution. The questions are based on the acronym SCAMPER, which stands for Substitute, Combine, Adapt, Modify, Put to another use, Eliminate, and Reverse. For each question, you can ask yourself how you can change or enhance your idea or solution. The goal is to generate new or alternative ideas or solutions from an existing one.
- Six Thinking Hats: This technique involves using different perspectives or modes of thinking to evaluate your ideas or solutions. The perspectives are based on the metaphor of wearing different coloured hats, which represent White (facts and data), Red (emotions and feelings), Black (criticism and risks), Yellow (optimism and benefits), Green (creativity and possibilities), and Blue (process and overview). For each perspective, you can ask yourself questions or make statements that reflect that mode of thinking. The goal is to consider your ideas or solutions from different angles and avoid biases or assumptions.

These are some examples of ideation and brainstorming techniques that you can use for the ASL Alphabet Image Recognition project.



4. Requirement Analysis

4.1 Functional Requirements

The functional requirements for the ASL Alphabet Image Recognition project are:

- The web application should allow the user to capture video from their webcam and display it on the screen.
- The web application should allow the user to select a mode of operation, either translation or learning.
- In the translation mode, the web application should use the machine learning model to recognize the ASL alphabet from the video stream and display the corresponding text on the screen.
- In the learning mode, the web application should use the machine learning model to recognize the ASL alphabet from the video stream and provide feedback to the user on their accuracy and performance.
- In the learning mode, the web application should also provide users with tutorials, quizzes, and games to help them learn and practice the ASL alphabet.
- The web application should store the user's data, such as name, email, preferences, progress, and scores, in the database.
- The web application should provide the user with a user-friendly interface, clear navigation, and a consistent design.

4.2 Non-Functional Requirements

The non-functional requirements for the ASL Alphabet Image Recognition project are:

- The web application should be compatible with different browsers, devices, and operating systems.
- The web application should be secure and protect the user's data and privacy.
- The web application should be reliable and handle errors and exceptions gracefully.
- The web application should be scalable and handle multiple users and requests simultaneously.
- The web application should be maintainable and easy to update and modify.
- The web application should be user-friendly and intuitive, and follow the web accessibility guidelines.
- The web application should be responsive and fast, and provide feedback and confirmation to the user.

- The web application should be testable and verifiable, and meet the quality standards and specifications.

5. Project Design

5.1 Data Flow Diagrams & User Stories

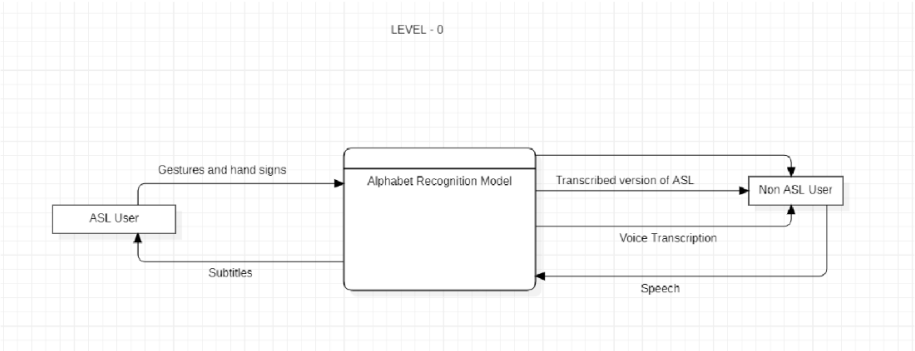
A DFD diagram is a way of representing the flow of data through a process or a system, such as a software application or an information system. A DFD diagram shows the movement of data between different components or entities, such as processes, data stores, and external sources or sinks. A DFD diagram also shows the inputs and outputs of each component or entity, and the transformations that occur on the data.

A DFD diagram can help you understand how a system works, what data is involved, and how the data is processed and stored. A DFD diagram can also help you identify potential problems, improve efficiency, and design better solutions. A DFD diagram can be used for various purposes, such as analysis, design, documentation, or communication.

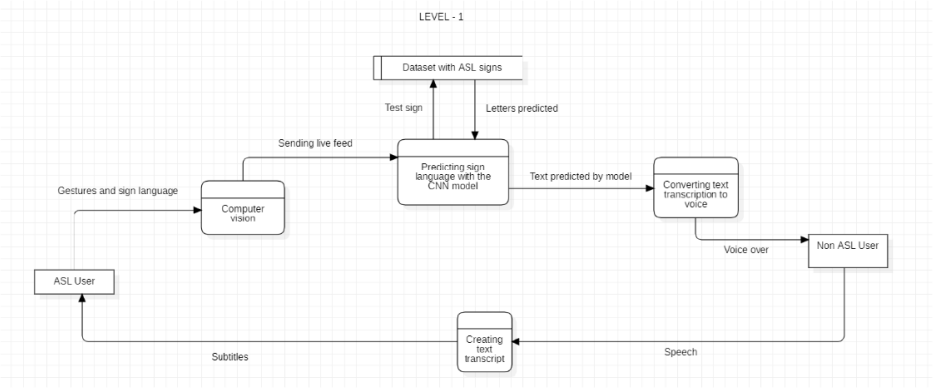
A DFD diagram can be organized into different levels of detail, depending on the complexity and purpose of the system. A DFD diagram can start with a high-level overview of the system, which is called a context diagram or a level 0 diagram. A context diagram shows the system as a single process, with its inputs and outputs from and to external sources or sinks. A context diagram can be further decomposed into more detailed diagrams, which are called level 1 diagrams. A level 1 diagram shows the main processes or functions of the system, and how they interact with each other and with the external sources or sinks. A level 1 diagram can be further decomposed into more detailed diagrams, called level 2 diagrams, and so on.

Data Flow Diagram

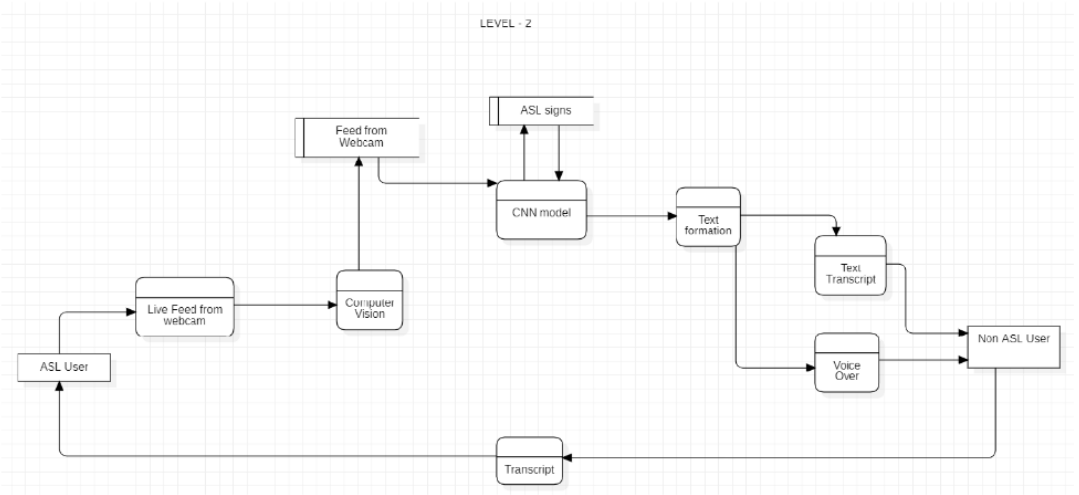
Level 0:



Level 1:



Level 2:



User Story

The user story for the ASL Alphabet Image Recognition project is:

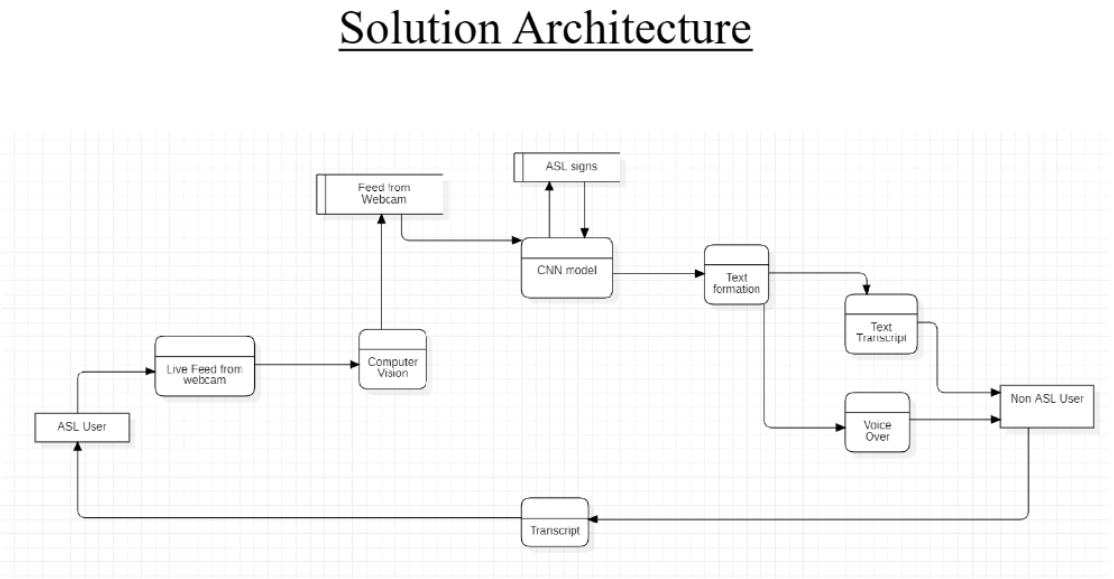
As a deaf user, I want to use the web application to translate my hand signs into text so that I can communicate with hearing people who do not know ASL.

Some additional user stories are

- As a hearing user, I want to use the web application to translate text into hand sign images so that I can communicate with deaf people who use ASL.
- As a deaf user, I want to use the web application to learn and practice the ASL alphabet, so that I can improve my sign language skills and confidence.
- As a hearing user, I want to use the web application to learn and practice the ASL alphabet, so that I can understand and appreciate the deaf culture and language.

5.2 Solution Architecture

Solution architecture is the process and outcome of designing and describing a specific solution that meets the requirements and goals of a business problem, within the context and constraints of the enterprise architecture. Solution architecture involves selecting and integrating the appropriate components of business, information, application, and technology architectures and defining the interfaces, interactions, and dependencies among them. Solution architecture also provides a roadmap and a plan for implementing, testing and deploying the solution. Solution architecture aims to ensure that the solution is aligned with the enterprise vision, strategy, and standards, as well as to deliver value, quality, and performance to the stakeholders and users.



6. Project Planning and Scheduling

6.1 Technical Architecture

the process and outcome of designing and describing a specific solution that meets the requirements and goals of a business problem, within the context and constraints of the enterprise architecture. Technical architecture involves selecting and integrating the appropriate components of business, information, application, and technology architectures and defining the interfaces, interactions, and dependencies among them. Technical architecture also provides a roadmap and a plan for implementing, testing and deploying the solution. Technical architecture aims to ensure that the solution is aligned with the enterprise vision, strategy, and standards, as well as to deliver value, quality, and performance to the stakeholders and users.

S No	Component	Description	Technology
1.	User Interface	Allows users to interact with the ASL recognition system. Includes elements for webcam access, video display, and interaction with the system	HTML, CSS, JavaScript , react
2.	Application Logic-1	It represents the core logic of the ASL recognition system. It involves handling user interactions, capturing video frames, and initiating the ASL recognition process	Python (Flask)
3.	Application Logic-2	This component of the application logic may include further processing of video frames and interaction with the ASL recognition model. It extends the core functionality.	Python, JavaScript
4.	Application Logic-3	additional processing or user-specific functionality, such as managing user profiles or maintaining session data.	Python, JavaScript
5.	Database	To store user live feed for processing etc	
6.	File Storage	To store all the essential components	Local Filesystem
7.	External API-1	An external API used for converting recognized ASL signs into voice output (Text-to-Speech, TTS) to enhance user communication.	gTTS/ IBM Watson Text to Speech
8.	Machine Learning Model	It is responsible for recognizing ASL signs from video frames	CNN(Convolutional Neural Network), Tensorflow
9.	Infrastructure (Server / Cloud)	Local Server Configuration	Local

6.2 Sprint Planning & Estimation

Sprint planning and estimation are two related activities that are part of the scrum framework, a popular agile methodology for software development. Sprint planning is an event in the scrum that defines what can be delivered in the upcoming sprint and how that work will be achieved. Sprint estimation is the process of predicting the effort and time required to complete the tasks in the sprint backlog.

Sprint planning involves the collaboration of the whole scrum team, which consists of the product owner, the development team, and the scrum master. The product owner describes the objective or goal of the sprint and what backlog items contribute to that goal. The

development team decides what can be done in the coming sprint and how they will do it. The scrum master facilitates the event and ensures that the team follows the scrum principles and values. The sprint planning session usually lasts for a maximum of eight hours for a four-week sprint or proportionally less for shorter sprints.

Sprint estimation involves the use of various techniques and methods to assess the complexity, risk, and effort of the tasks in the sprint backlog. The most common unit of measurement for sprint estimation is the story point, which is a relative and abstract value that represents the size of a task. The development team assigns story points to each task based on their experience, knowledge, and judgment. Some of the popular techniques for sprint estimation are planning poker, affinity mapping, t-shirt sizing, and dot voting. The purpose of sprint estimation is not to achieve perfect accuracy, but to provide a reasonable forecast that can help the team plan and deliver the sprint goal.

Sprint	Functional Requirement (Epic)	User Story Number	User Story/Task	Story Points	Priority
Sprint 1	Data Collection	USN-1	Collect ASL dataset	2	High
Sprint 1	Model Development	USN-2	Develop ASL recognition model	8	High
Sprint 1	Model Development	USN-3	Train and test ASL model	6	High
Sprint 2	User Interface	USN-4	UI development	8	High
Sprint 2	Webcam Integration	USN-5	Integrate webcam for video capture	3	Medium
Sprint 3	Text-to-Speech	USN-6	Integrate TTS service	2	Medium
Sprint 3	Testing & Optimization	USN-7	Testing and optimization	3	High

6.3 Sprint Delivery Schedule

A sprint delivery schedule is a document or tool that shows the planned work items and features that a team or a group of teams will deliver in a series of sprints. A sprint is a short time-boxed period, usually lasting one to four weeks, in which a team works on a set of tasks or user stories that contribute to a product or a solution. A sprint delivery schedule helps coordinate and align the efforts of multiple teams working on a large or complex project and provides visibility and feedback to the stakeholders and users.

The sprint delivery schedule is typically created at the beginning of a sprint and is updated throughout the sprint as the team progresses through the work. The schedule includes the following information:

- The work items and features that will be delivered in the sprint
- The tasks that need to be completed to deliver each work item or feature
- The team members who are responsible for each task
- The estimated time it will take to complete each task
- The dependencies between tasks

- The risks and issues that could impact the sprint

Delivery schedule is a valuable tool for teams working on agile projects. It helps teams to stay on track, communicate effectively, and deliver high-quality work.

Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed (as on Planned End Date)	Sprint Release Date (Actual)
Sprint-1	16	10 Days	12 Oct 2023	22 Oct 2023	16	22 Oct 2023
Sprint-2	11	8 Days	17 Oct 2023	25 Oct 2023	8	27 Oct 2023
Sprint-3	5	6 Days	25 Oct 2023	1 Nov 2023	2	04 Nov 2023

7. Coding and Solutions

7.1 Feature 1: Real-time text formation:

This feature involves the real-time prediction of the ASL signs.

```
def gen_frames():
    global message
    global last_prediction
    while True:
      # Capture frame-by-frame
      ret, frame = cap.read()

      # Target area where the hand gestures should be.
      cv2.rectangle(frame, (0, 0), (CROP_SIZE, CROP_SIZE), (0, 255, 0), 3)

      # Preprocessing the frame before input to the model.
      cropped_image = frame[0:CROP_SIZE, 0:CROP_SIZE]
      resized_frame = cv2.resize(cropped_image, (IMAGE_SIZE, IMAGE_SIZE))
      reshaped_frame = (np.array(resized_frame)).reshape((1, IMAGE_SIZE, IMAGE_SIZE, 3))
      frame_for_model = data_generator.standardize(np.float64(reshaped_frame))

      # Predicting the frame.
      prediction = np.array(model.predict(frame_for_model))
      predicted_class = classes[prediction.argmax()]      # Selecting the max confidence index.

      frame = cv2.flip(frame, 1)
      # Preparing output based on the model's confidence.
      prediction_probability = prediction[0, prediction.argmax()]
```

In the above function, a frame is created in the video within which the ASL signs will be recognized and predicted. The frames obtained are then preprocessed and given as input into the model. The model then predicts the probability of the ASL sign.

```
if prediction_probability > 0.6:
    # High confidence.
    cv2.putText(frame, '          {} - {:.2f}%'.format(predicted_class, prediction_probability * 100),
                (10, 450), 1, 2, (255, 255, 0), 2, cv2.LINE_AA)
    # Append the predicted class to the message if it's different from the last prediction
    if predicted_class != last_prediction:
        if predicted_class == 'space':
            message += ' '
        elif predicted_class=='nothing':
            message +=""
        else:
            message += predicted_class
        last_prediction = predicted_class
elif prediction_probability > 0.2 and prediction_probability <= 0.6:
    # Low confidence.
    cv2.putText(frame, '          Maybe {}... - {:.2f}%'.format(predicted_class, prediction_probability * 100),
                (10, 450), 1, 2, (0, 255, 255), 2, cv2.LINE_AA)
else:
    # No confidence.
    cv2.putText(frame, classes[-2], (10, 450), 1, 2, (255, 255, 0), 2, cv2.LINE_AA)

# Encode the frame as a JPEG image
_, buffer = cv2.imencode('.jpg', frame)
frame = buffer.tobytes()
```

The prediction probability calculated is then used to confirm the letters. If the probability is greater than 60% the corresponding letter is added to the sentence, it won't be added. The ‘cv2.putText()’ function shows the prediction probability under the frame.

```
# Encode the frame as a JPEG image
_, buffer = cv2.imencode('.jpg', frame)
frame = buffer.tobytes()

# Yield the frame as a Flask response
yield (b'--frame\r\n'
       b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n')

# Emit the message to the client
socketio.emit('message', message)
```

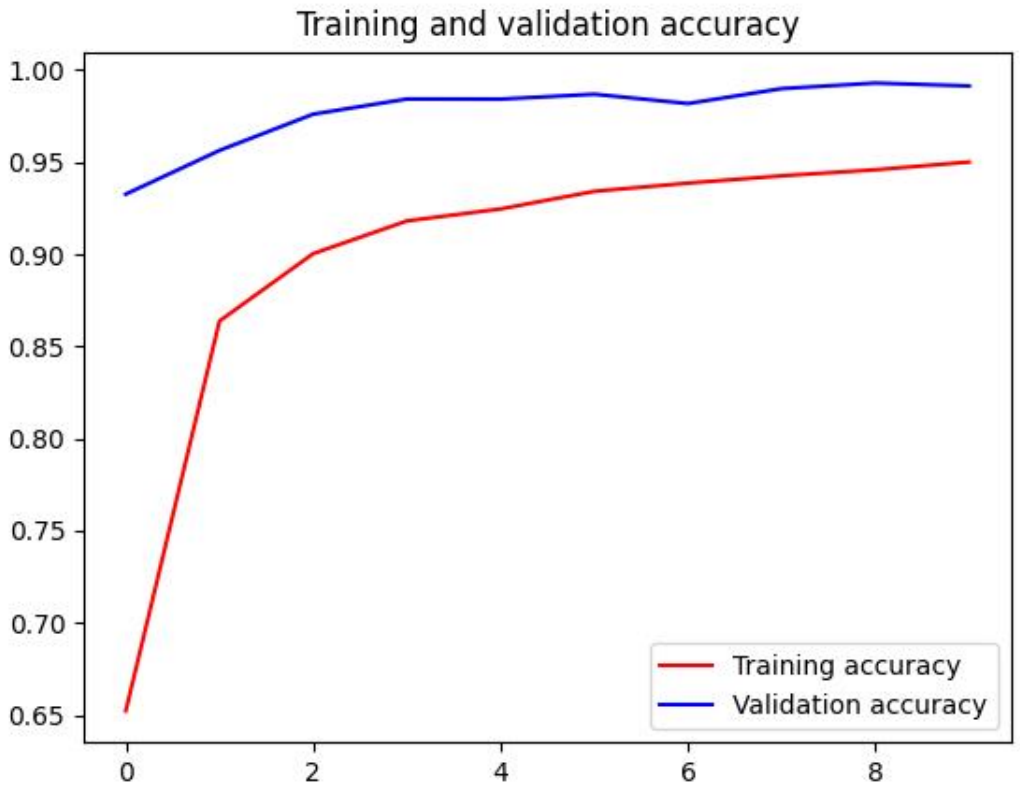
The code above is used for video streaming over a network. ‘cv2.imencode()’ encodes the video frames into JPEG and the results are then stored in _buffer. The ‘tobytes’ function is used to convert the buffer into a byte array. Finally the socketio.emit function is used to send the HTTP response to the client over a Socket.IO connection.

7.2 Feature 2: Text-to-Speech conversion:

```
// Add event listener to the "Speak" button
document.getElementById('speakButton').addEventListener('click', function() {
    // Get the message from the paragraph with id 'message'
    var message = document.getElementById('message').innerHTML;

    // Create a new SpeechSynthesisUtterance object with the message
    var utterance = new SpeechSynthesisUtterance(message);

    // Pass the utterance to speechSynthesis.speak()
    window.speechSynthesis.speak(utterance);
```

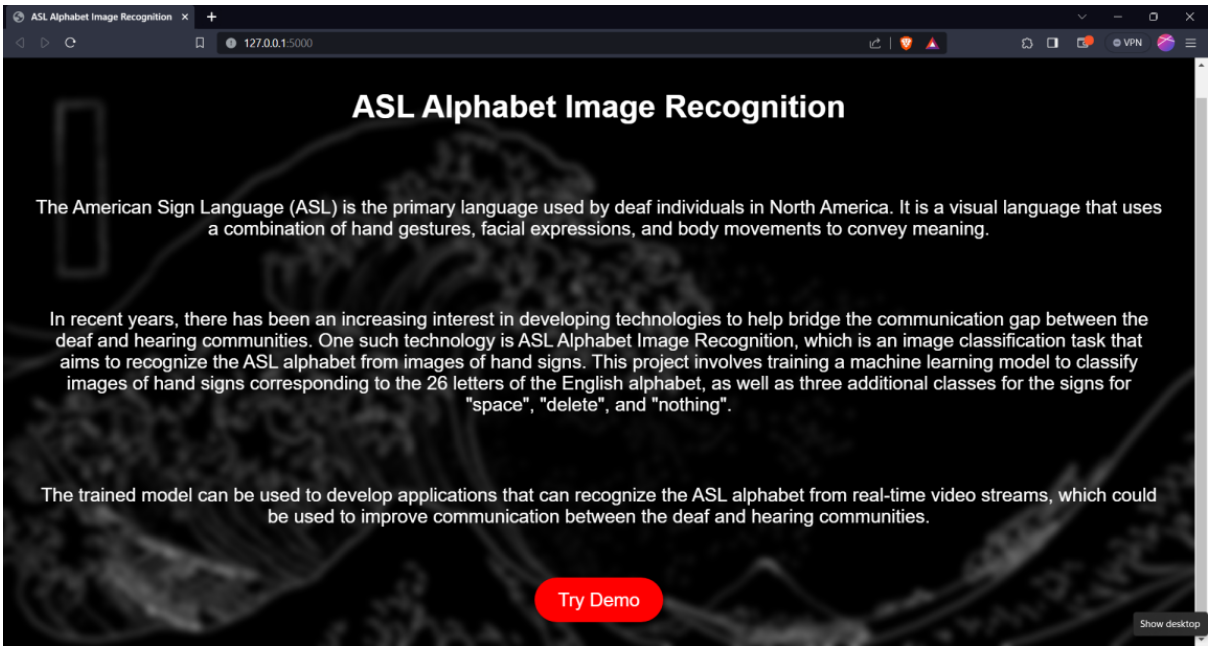
This code snippet involves a predefined JavaScript API ‘SpeechSynthesisUtterance’ which is used for controlling and managing text-to-speech functionality in web applications. The message which has been produced by the model is then passed into the function which converts the text message to voice.

8. Performance Testing

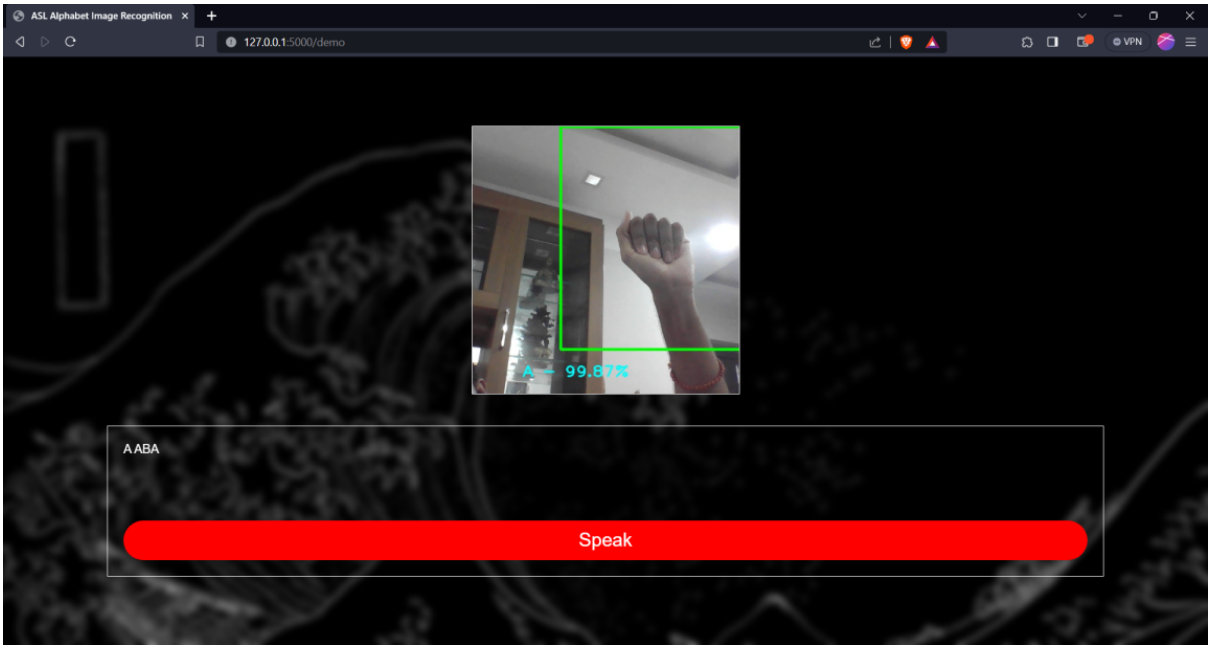


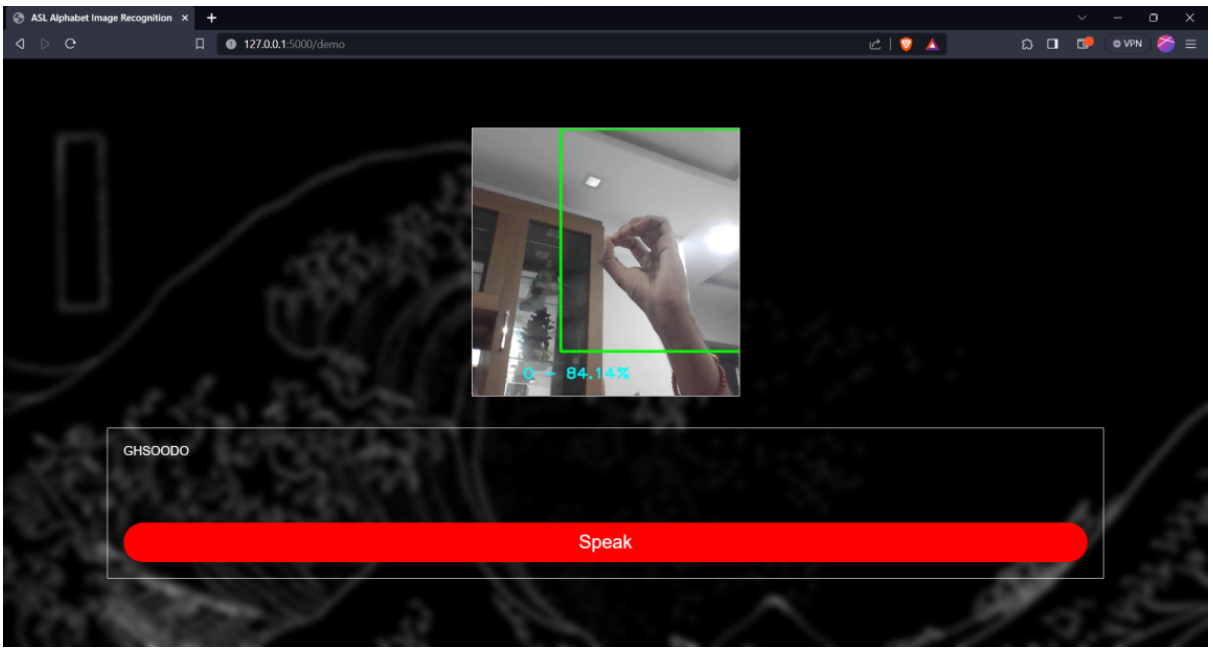
9. Results

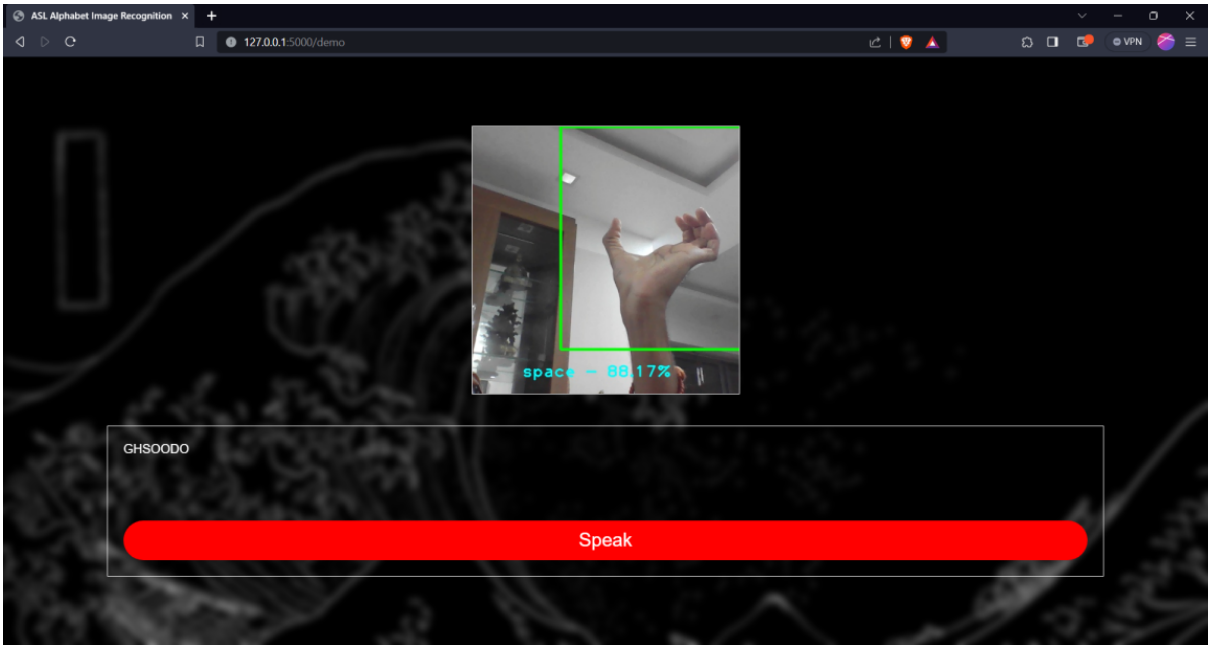
Home page:



Demo Page:







10. Advantages and Disadvantages

10.1 Advantages:

- The project has the capability to process real-time video so it allows for real-time interaction.

- The project can be deployed and integrated into various applications such as video conferencing and assistive communication devices making it versatile.
- It helps bridge the communication gap between deaf and hearing communities, allowing for more effective and inclusive communication.
- The text-to-speech feature also allows for the deaf communities to communicate with the blind communities using sign language.
- CNNs simplify the learning process by automatically figuring out important features from the training data. This makes it easier to train the model without needing to manually define specific characteristics of ASL hand signs.

10.2 Disadvantages:

- The model may face challenges in accurately recognizing gestures in certain lighting conditions, backgrounds, or variations in hand movements, potentially leading to misinterpretation.
- Loading a pre-trained model during the application's initialization might introduce a delay, impacting the user experience, especially on slower devices or network connections.
- Real-time video processing can be resource-intensive, requiring significant processing power. This may limit the application's performance on less powerful devices.
- The performance of the application may be affected by the quality of the webcam. Lower-quality webcams might result in decreased accuracy and reliability.
- The model only recognizes alphabets, which are usually not used frequently by ASL users. They mostly use different gestures and phrases while communicating with each other.

11. Conclusion

The ASL Alphabet Image Recognition project successfully bridges the communication gap between deaf individuals using American Sign Language (ASL) and those unfamiliar with it. Starting with a robust dataset and employing preprocessing techniques, the project utilized a Convolutional Neural Network (CNN) model, powered by TensorFlow, to achieve accurate hand symbol classification.

The web application deployment, featuring a user-friendly, allows real-time ASL recognition from webcam feeds. Beyond technical accomplishments, the project's core purpose is to enhance inclusivity and communication accessibility. Its potential impact spans education, public services, and personal interactions, exemplifying the significance of technology in fostering understanding.

As an assistive tool, the ASL Alphabet Image Recognition project contributes to a more connected society by providing a practical solution to the lack of ASL familiarity. This project stands as a testament to technology's power in promoting inclusivity and overcoming communication barriers, paving the way for a more connected and understanding world.

12. Future Scope

The future scope of the ASL Image Recognition project is vast. It could revolutionize communication for the deaf community by providing real-time translation of ASL into text or speech. This technology could be integrated into various platforms, enhancing accessibility in digital communication, education, and entertainment. It could also aid in learning ASL, acting as a digital tutor. Furthermore, advancements in AI and machine learning could improve the accuracy and speed of recognition. Ultimately, this project has the potential to bridge the communication gap, fostering inclusivity and understanding.

- Digital Communication: By integrating this technology into digital platforms, we can make online communication more accessible for the deaf community.

For instance, video conferencing tools could use this technology to provide real-time captions for sign language.

- Education: This technology could be used to develop educational tools that help people learn ASL. It could provide instant feedback to learners, improving their sign language proficiency.
- Entertainment: In the entertainment industry, real-time ASL translation could make content more accessible. For example, it could be used to provide sign language interpretations for live events or broadcasts.
- Artificial Intelligence and Machine Learning: As AI and ML technologies advance, the accuracy and speed of ASL recognition could be significantly improved. This would make the translations more reliable, providing a better user experience.
- Inclusivity: Above all, this project could play a crucial role in fostering inclusivity. By making ASL more accessible, we can bridge the communication gap between the deaf community and the rest of society, promoting greater understanding and empathy.

13. Appendix:

13.1 Source Code:

Model Building Code:

```
!pip install kaggle
```
[2] Python

```
Requirement already satisfied: kaggle in /usr/local/lib/python3.10/dist-packages (1.5.16)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.10/dist-packages (from kaggle) (1.16.0)
Requirement already satisfied: certifi in /usr/local/lib/python3.10/dist-packages (from kaggle) (2023.7.22)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.10/dist-packages (from kaggle) (2.8.2)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from kaggle) (2.31.0)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from kaggle) (4.66.1)
Requirement already satisfied: python-slugify in /usr/local/lib/python3.10/dist-packages (from kaggle) (8.0.1)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.10/dist-packages (from kaggle) (2.0.7)
Requirement already satisfied: bleach in /usr/local/lib/python3.10/dist-packages (from kaggle) (6.1.0)
Requirement already satisfied: webencodings in /usr/local/lib/python3.10/dist-packages (from bleach->kaggle) (0.5.1)
Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python3.10/dist-packages (from python-slugify->kaggle) (1.3)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->kaggle) (3.3.1)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->kaggle) (3.4)
```

```
!mkdir ~/.kaggle
```
[3] Python

```
!cp kaggle.json ~/.kaggle
```
[5] Python

Setting permission to owner alone

```
!chmod 600 ~/.kaggle/kaggle.json
```
[6] Python

Downloading dataset from kaggle using API command

```
!kaggle datasets download -d grassknoted/asl-alphabet
```
[7] Python

```
Downloading asl-alphabet.zip to /content
100% 1.02G/1.03G [00:13<00:00, 117MB/s]
100% 1.03G/1.03G [00:13<00:00, 84.4MB/s]
```

```
!unzip asl-alphabet.zip -d asl-alphabet
```
[8] Python

```
Streaming output truncated to the last 5000 lines.
  inflating: asl-alphabet/asl_alphabet_train/asl_alphabet_train/nothing/nothing19.jpg
  inflating: asl-alphabet/asl_alphabet_train/asl_alphabet_train/nothing/nothing190.jpg
  inflating: asl-alphabet/asl_alphabet_train/asl_alphabet_train/nothing/nothing1900.jpg
  inflating: asl-alphabet/asl_alphabet_train/asl_alphabet_train/nothing/nothing1901.jpg
  inflating: asl-alphabet/asl_alphabet_train/asl_alphabet_train/nothing/nothing1902.jpg
  inflating: asl-alphabet/asl_alphabet_train/asl_alphabet_train/nothing/nothing1903.jpg
  inflating: asl-alphabet/asl_alphabet_train/asl_alphabet_train/nothing/nothing1904.jpg
  inflating: asl-alphabet/asl_alphabet_train/asl_alphabet_train/nothing/nothing1905.jpg
  inflating: asl-alphabet/asl_alphabet_train/asl_alphabet_train/nothing/nothing1906.jpg
  inflating: asl-alphabet/asl_alphabet_train/asl_alphabet_train/nothing/nothing1907.jpg
  inflating: asl-alphabet/asl_alphabet_train/asl_alphabet_train/nothing/nothing1908.jpg
  inflating: asl-alphabet/asl_alphabet_train/asl_alphabet_train/nothing/nothing1909.jpg
  inflating: asl-alphabet/asl_alphabet_train/asl_alphabet_train/nothing/nothing191.jpg
  inflating: asl-alphabet/asl_alphabet_train/asl_alphabet_train/nothing/nothing1910.jpg
  inflating: asl-alphabet/asl_alphabet_train/asl_alphabet_train/nothing/nothing1911.jpg
  inflating: asl-alphabet/asl_alphabet_train/asl_alphabet_train/nothing/nothing1912.jpg
  inflating: asl-alphabet/asl_alphabet_train/asl_alphabet_train/nothing/nothing1913.jpg
  inflating: asl-alphabet/asl_alphabet_train/asl_alphabet_train/nothing/nothing1914.jpg
  inflating: asl-alphabet/asl_alphabet_train/asl_alphabet_train/nothing/nothing1915.jpg
```

```python
# importing main modules
import warnings
import os
import glob
import cv2
import numpy as np
import pandas as pd
import gc
import string
import time
import random
import imutils
from PIL import Image
from tqdm import tqdm
tqdm.pandas()

#Visualization modules
import matplotlib
import matplotlib.pyplot as plt
import plotly
import plotly.graph_objects as go
import plotly.express as px
from plotly.subplots import make_subplots
from sklearn.manifold import TSNE

# Modules for model building
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import VGG16
from tensorflow.keras.layers import Dense, Flatten, Dropout
from keras.models import load_model, Model
from keras.optimizers import Adam
from keras.callbacks import ModelCheckpoint, EarlyStopping
```

```python
class CFG:
  batch_size = 64
  img_height = 64
  img_width = 64
  epochs = 10
  num_classes = 29
  img_channels = 3

def seed_everything(seed: int):
  random.seed(seed)
  os.environ["PYTHONHASHSEED"] = str(seed)
  np.random.seed(seed)
  tf.random.set_seed(seed)
```

Python

```python
# Labels
TRAIN_PATH = "/content/asl-alphabet/asl_alphabet_train/asl_alphabet_train"
labels = []
alphabet = list(string.ascii_uppercase)
labels.extend(alphabet)
labels.extend(["del", "nothing", "space"])
print(labels)
```

Python

```
['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', 'nothing', 'space']
```

Creating Metadata

```
list_path = []
list_labels = []
for label in labels:
  label_path = os.path.join(TRAIN_PATH, label, "*")
  image_files = glob.glob(label_path)

  sign_label = [label]*len(image_files)

  list_path.extend(image_files)
  list_labels.extend(sign_label)

metadata = pd.DataFrame({"image_path": list_path, "label":list_labels})
```

[13] Python

Split Dataset to Train 0.7, Val 0.15, and Test 0.15

```
x_train, x_test, y_train, y_test = train_test_split(metadata['image_path'], metadata['label'], test_size=0.15, random_state=0, shuffle=True, stratify=metadata[
data_train = pd.DataFrame({"image_path":x_train, "label":y_train})
data_test = pd.DataFrame({"image_path":x_test, "label":y_test})

x_train, x_val, y_train, y_val = train_test_split(data_train["image_path"], data_train['label'], test_size=0.15/0.70, random_state=0, shuffle=True, stratify=da
data_val = pd.DataFrame({"image_path":x_val, "label":y_val})
data_train = pd.DataFrame({"image_path":x_train, "label":y_train})
```

[14] Python

Data augmentation

```
def data_augmentation():
  datagen =ImageDataGenerator(rescale=1/255.,)
  train_generator = datagen.flow_from_dataframe(data_train, directory="./",
                    x_col="image_path", y_col="label", class_mode="categorical",
                    batch_size=CFG.batch_size, target_size=(CFG.img_height, CFG.img_width),)

  validation_generator = datagen.flow_from_dataframe(data_val, directory="./",
                    x_col="image_path", y_col="label", class_mode="categorical",
                    batch_size=CFG.batch_size, target_size=(CFG.img_height, CFG.img_width),)

  test_generator = datagen.flow_from_dataframe(data_test, directory="./",
                    x_col="image_path", y_col="label", class_mode="categorical",
                    batch_size=CFG.batch_size, target_size=(CFG.img_height, CFG.img_width),)

  return train_generator, validation_generator, test_generator
```

[15]

```
seed_everything(2023)
train_generator, validation_generator, test_generator = data_augmentation()
```

[16]

```
Found 58103 validated image filenames belonging to 29 classes.
Found 15847 validated image filenames belonging to 29 classes.
Found 13050 validated image filenames belonging to 29 classes.
```

Model Building

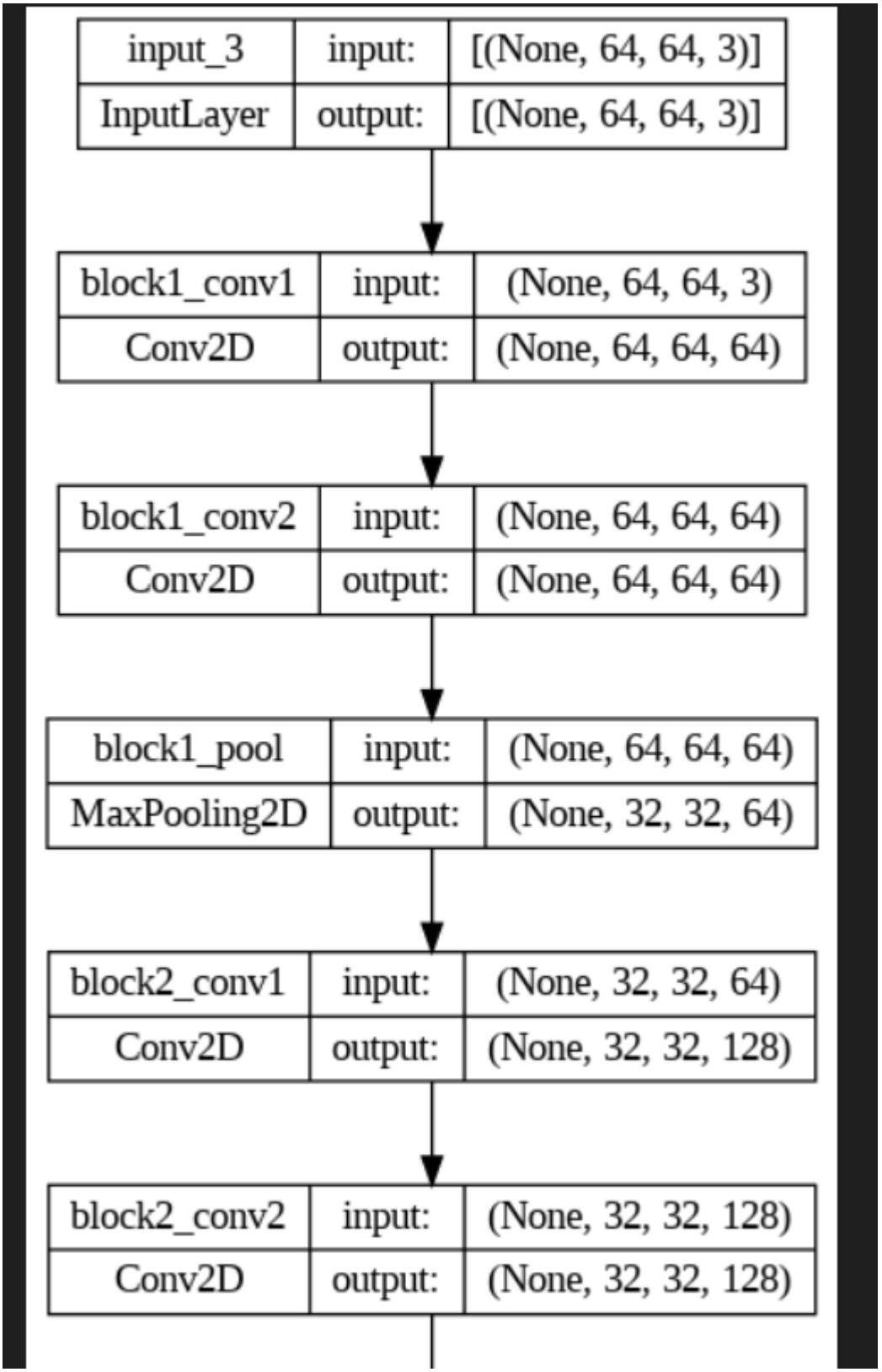
```
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(64, 64, 3))

for layer in base_model.layers:
  layer.trainable = False

x = base_model.output
x = Flatten()(x)
x = Dense(512, activation='relu')(x)
x = Dropout(0.5)(x)
x = Dense(512, activation='relu')(x)
x = Dropout(0.5)(x)
predictions = Dense(29, activation='softmax')(x)

model = Model(inputs = base_model.input, outputs = predictions)
display(model.summary())
display(tf.keras.utils.plot_model(model, to_file='vgg16.png', show_shapes=True))
```

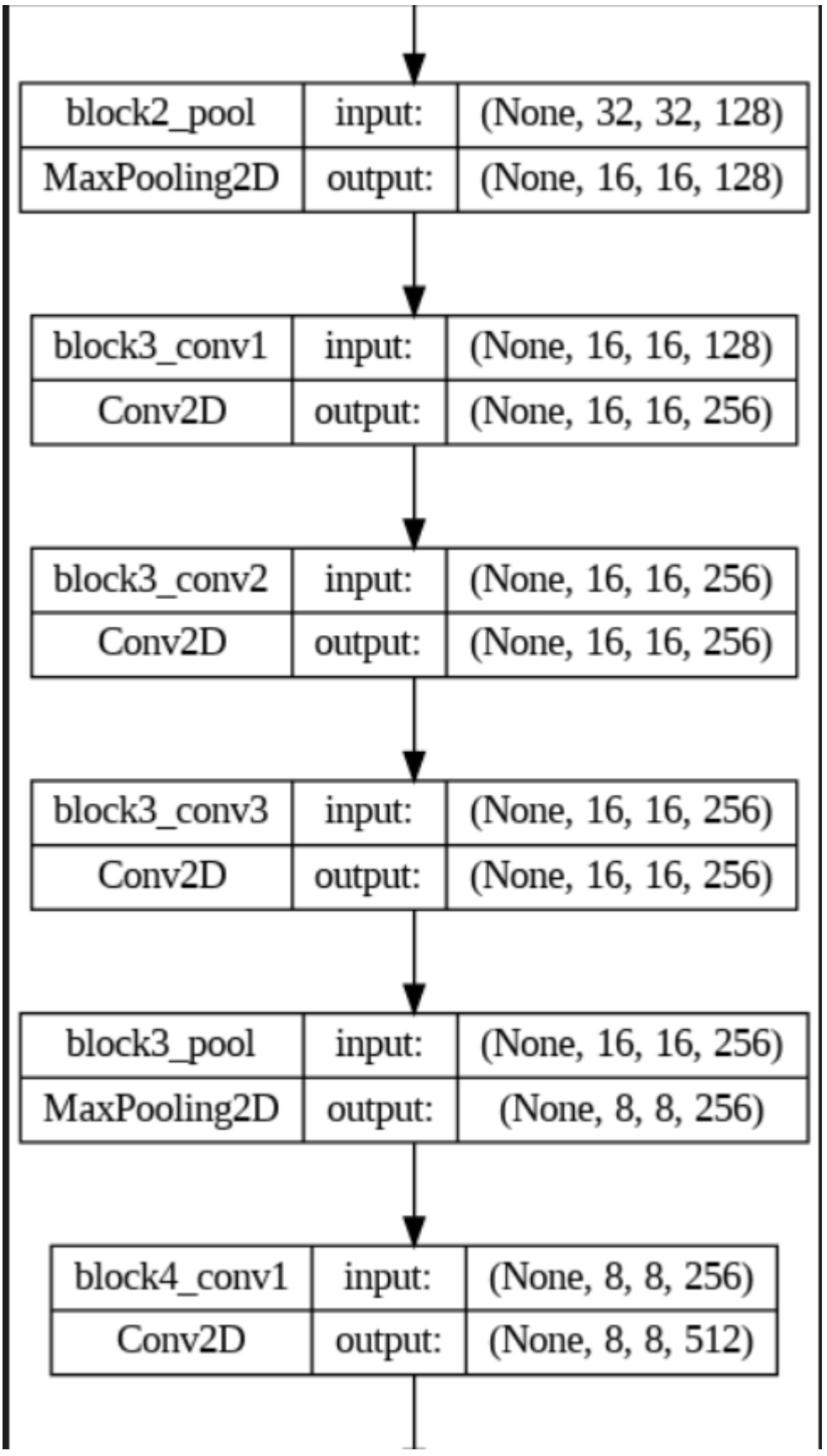
[23]

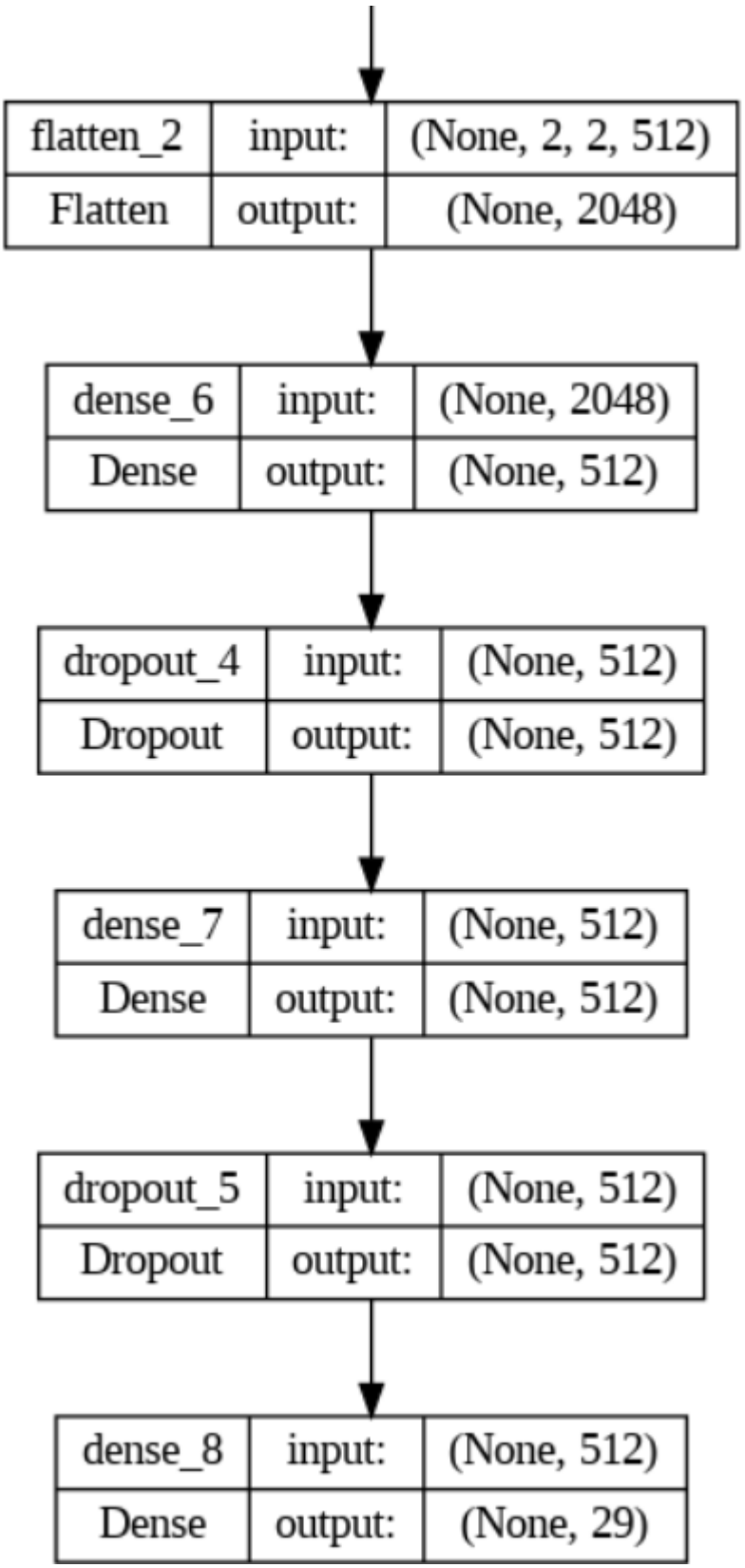
```
Model: "model_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, 64, 64, 3)]       0

 block1_conv1 (Conv2D)       (None, 64, 64, 64)        1792

 block1_conv2 (Conv2D)       (None, 64, 64, 64)        36928

 block1_pool (MaxPooling2D)  (None, 32, 32, 64)        0

 block2_conv1 (Conv2D)       (None, 32, 32, 128)       73856

 block2_conv2 (Conv2D)       (None, 32, 32, 128)       147584
```

```
 block2_conv2 (Conv2D)       (None, 32, 32, 128)       147584

 block2_pool (MaxPooling2D)  (None, 16, 16, 128)       0

 block3_conv1 (Conv2D)       (None, 16, 16, 256)       295168

 block3_conv2 (Conv2D)       (None, 16, 16, 256)       590080

 block3_conv3 (Conv2D)       (None, 16, 16, 256)       590080

 block3_pool (MaxPooling2D)  (None, 8, 8, 256)         0
...
Total params: 16041309 (61.19 MB)
Trainable params: 1326621 (5.06 MB)
Non-trainable params: 14714688 (56.13 MB)
_________________________________________________________________
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...


input_3 | input: | [(None, 64, 64, 3)]
InputLayer | output: | [(None, 64, 64, 3)]
block1_conv1 | input: | (None, 64, 64, 3)
Conv2D | output: | (None, 64, 64, 64)
block1_conv2 | input: | (None, 64, 64, 64)
Conv2D | output: | (None, 64, 64, 64)
block1_pool | input: | (None, 64, 64, 64)
MaxPooling2D | output: | (None, 32, 32, 64)
block2_conv1 | input: | (None, 32, 32, 64)
Conv2D | output: | (None, 32, 32, 128)
block2_conv2 | input: | (None, 32, 32, 128)
Conv2D | output: | (None, 32, 32, 128)


block2_pool
input:
(None, 32, 32, 128)
MaxPooling2D
output:
(None, 16, 16, 128)
block3_conv1
input:
(None, 16, 16, 128)
Conv2D
output:
(None, 16, 16, 256)
block3_conv2
input:
(None, 16, 16, 256)
Conv2D
output:
(None, 16, 16, 256)
block3_conv3
input:
(None, 16, 16, 256)
Conv2D
output:
(None, 16, 16, 256)
block3_pool
input:
(None, 16, 16, 256)
MaxPooling2D
output:
(None, 8, 8, 256)
block4_conv1
input:
(None, 8, 8, 256)
Conv2D
output:
(None, 8, 8, 512)


flatten_2 | input: | (None, 2, 2, 512)
Flatten | output: | (None, 2048)
dense_6 | input: | (None, 2048)
Dense | output: | (None, 512)
dropout_4 | input: | (None, 512)
Dropout | output: | (None, 512)
dense_7 | input: | (None, 512)
Dense | output: | (None, 512)
dropout_5 | input: | (None, 512)
Dropout | output: | (None, 512)
dense_8 | input: | (None, 512)
Dense | output: | (None, 29)

```
#compile and train the model
model.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossentropy', metrics=["accuracy"])

#callbacks
checkpoint = ModelCheckpoint('asl_vgg16_best_weights.h5', save_best_only=True, monitor='val_loss', mode = 'min')
```

```
for batch in train_generator:
  x_batch, y_batch = batch
  print("batch shape: ", x_batch.shape)
  print("data type: ", x_batch.dtype)
  break
```

```
batch shape:  (64, 64, 64, 3)
data type:  float32
```

```
input_layer = model.layers[0]
input_shape = input_layer.input_shape
input_shape
```

```
[(None, 64, 64, 3)]
```

```
history = model.fit(train_generator, steps_per_epoch=train_generator.samples // CFG.batch_size,
                    epochs=CFG.epochs, validation_data=validation_generator,
                    validation_steps=validation_generator.samples // CFG.batch_size, callbacks=[checkpoint])
```

```
Epoch 1/10
907/907 [==============================] - 3153s 3s/step - loss: 1.0806 - accuracy: 0.6581 - val_loss: 0.2242 - val_accuracy: 0.9367
/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
  saving_api.save_model(
Epoch 2/10
907/907 [==============================] - 3076s 3s/step - loss: 0.3976 - accuracy: 0.8658 - val_loss: 0.1149 - val_accuracy: 0.9670
Epoch 3/10
907/907 [==============================] - 3080s 3s/step - loss: 0.2874 - accuracy: 0.9012 - val_loss: 0.0819 - val_accuracy: 0.9768
Epoch 4/10
907/907 [==============================] - 3041s 3s/step - loss: 0.2427 - accuracy: 0.9173 - val_loss: 0.0706 - val_accuracy: 0.9811
Epoch 5/10
907/907 [==============================] - 3087s 3s/step - loss: 0.2131 - accuracy: 0.9266 - val_loss: 0.0773 - val_accuracy: 0.9749
Epoch 6/10
907/907 [==============================] - 3059s 3s/step - loss: 0.1950 - accuracy: 0.9333 - val_loss: 0.0560 - val_accuracy: 0.9832
Epoch 7/10
907/907 [==============================] - 3072s 3s/step - loss: 0.1793 - accuracy: 0.9400 - val_loss: 0.0485 - val_accuracy: 0.9872
Epoch 8/10
907/907 [==============================] - 3108s 3s/step - loss: 0.1630 - accuracy: 0.9435 - val_loss: 0.0497 - val_accuracy: 0.9839
Epoch 9/10
907/907 [==============================] - 3065s 3s/step - loss: 0.1627 - accuracy: 0.9449 - val_loss: 0.0421 - val_accuracy: 0.9883
Epoch 10/10
907/907 [==============================] - 3058s 3s/step - loss: 0.1547 - accuracy: 0.9482 - val_loss: 0.0341 - val_accuracy: 0.9903
```

Model Evaluation

```
scores = model.evaluate(test_generator)
print("%s: %.2f%%" % ("evaluate Test Accuracy", scores[1]*100))
```

```
204/204 [==============================] - 537s 3s/step - loss: 0.0325 - accuracy: 0.9928
evaluate Test Accuracy: 99.28%
```

```
# confusion matrix
fine_tuned_model = load_model("/content/asl_vgg16_best_weights.h5")
predictions = fine_tuned_model.predict(test_generator)

# Getting the labels form the generator
true_labels = test_generator.classes

# compute the confusion matrix using tf.math.confusion_matrix
confusion_matrix = tf.math.confusion_matrix(labels=true_labels, predictions=predictions.argmax(axis=1), num_classes=29)
```

```
204/204 [==============================] - 534s 3s/step
```

```
[34]
dense_model = Model(inputs=fine_tuned_model.inputs, outputs = fine_tuned_model.layers[-1].output)
dense_model.summary()
```

```
Model: "model_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, 64, 64, 3)]       0

 block1_conv1 (Conv2D)       (None, 64, 64, 64)        1792

 block1_conv2 (Conv2D)       (None, 64, 64, 64)        36928

 block1_pool (MaxPooling2D)  (None, 32, 32, 64)        0

 block2_conv1 (Conv2D)       (None, 32, 32, 128)       73856

 block2_conv2 (Conv2D)       (None, 32, 32, 128)       147584

 block2_pool (MaxPooling2D)  (None, 16, 16, 128)       0

 block3_conv1 (Conv2D)       (None, 16, 16, 256)       295168

 block3_conv2 (Conv2D)       (None, 16, 16, 256)       590080

 block3_conv3 (Conv2D)       (None, 16, 16, 256)       590080

 block3_pool (MaxPooling2D)  (None, 8, 8, 256)         0
...
Total params: 16041309 (61.19 MB)
Trainable params: 1326621 (5.06 MB)
Non-trainable params: 14714688 (56.13 MB)
```

```
# extracting features in dense layer

def dense_feature_prediction(img_path):
  img = load_img(img_path, target_size=(CFG.img_height, CFG.img_width))
  img = img_to_array(img)
  img = img/255.
  img = np.expand_dims(img, axis=0)
  dense_feature = dense_model.predict(img, verbose=0)[0]
  return dense_feature

reduction_data = pd.DataFrame()
for label in labels:
  label_data = data_test[data_test["label"]==label][:100]
  reduction_data = reduction_data.append(label_data)

reduction_data = reduction_data.reset_index(drop=True)
display(reduction_data)

dense_features = reduction_data["image_path"].progress_apply(dense_feature_prediction)
dense_features = pd.DataFrame.from_records(dense_features.values, index=dense_features.index)

#TSNE Dimnesional Reduction
tsne = TSNE(n_components=2, verbose=1, random_state=0, angle=.99, init='pca')
tsne_features = tsne.fit_transform(dense_features)
tsne_features = pd.DataFrame(tsne_features, columns=["tsne_feat_0", "tsne_feat_1"])
reduction_data[["tsne_feat_0", "tsne_feat_1"]] = tsne_features
reduction_data
```

Python

```
<ipython-input-36-e5245f22fcdb>:14: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat in
  reduction_data = reduction_data.append(label_data)
<ipython-input-36-e5245f22fcdb>:14: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat in
  reduction_data = reduction_data.append(label_data)
<ipython-input-36-e5245f22fcdb>:14: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat in
  reduction_data = reduction_data.append(label_data)
<ipython-input-36-e5245f22fcdb>:14: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat in
```

	image_path	label
0	/content/asl-alphabet/asl_alphabet_train/asl_a...	A
1	/content/asl-alphabet/asl_alphabet_train/asl_a...	A
2	/content/asl-alphabet/asl_alphabet_train/asl_a...	A
3	/content/asl-alphabet/asl_alphabet_train/asl_a...	A
4	/content/asl-alphabet/asl_alphabet_train/asl_a...	A
...	...	...
2895	/content/asl-alphabet/asl_alphabet_train/asl_a...	space
2896	/content/asl-alphabet/asl_alphabet_train/asl_a...	space
2897	/content/asl-alphabet/asl_alphabet_train/asl_a...	space
2898	/content/asl-alphabet/asl_alphabet_train/asl_a...	space
2899	/content/asl-alphabet/asl_alphabet_train/asl_a...	space

2900 rows × 2 columns

```
100%|██████████| 2900/2900 [06:38<00:00,  7.28it/s]
[t-SNE] Computing 91 nearest neighbors...
[t-SNE] Indexed 2900 samples in 0.001s...
[t-SNE] Computed neighbors for 2900 samples in 0.481s...
[t-SNE] Computed conditional probabilities for sample 1000 / 2900
[t-SNE] Computed conditional probabilities for sample 2000 / 2900
[t-SNE] Computed conditional probabilities for sample 2900 / 2900
[t-SNE] Mean sigma: 0.000000
[t-SNE] KL divergence after 250 iterations with early exaggeration: 45.463917
[t-SNE] KL divergence after 1000 iterations: 0.168743
```

	image_path	label	tsne_feat_0	tsne_feat_1
0	/content/asl-alphabet/asl_alphabet_train/asl_a...	A	14.028865	-8.589529
1	/content/asl-alphabet/asl_alphabet_train/asl_a...	A	11.814671	-10.658160
2	/content/asl-alphabet/asl_alphabet_train/asl_a...	A	11.806844	-10.914112
3	/content/asl-alphabet/asl_alphabet_train/asl_a...	A	14.966569	-9.648602
4	/content/asl-alphabet/asl_alphabet_train/asl_a...	A	11.296180	-11.579964
...	...	...	...	...
2895	/content/asl-alphabet/asl_alphabet_train/asl_a...	space	-37.516762	7.064789
2896	/content/asl-alphabet/asl_alphabet_train/asl_a...	space	-37.410606	6.375655
2897	/content/asl-alphabet/asl_alphabet_train/asl_a...	space	-35.307594	8.348761
2898	/content/asl-alphabet/asl_alphabet_train/asl_a...	space	-35.862339	11.372965
2899	/content/asl-alphabet/asl_alphabet_train/asl_a...	space	-40.113525	10.144936

Testing:

Loading and testing the model

Test - 1

```
#Load the saved model
model = tf.keras.models.load_model("/content/asl_vgg16_best_weights.h5")
# Load the test image
image_path = "/content/asl-alphabet/asl_alphabet_test/asl_alphabet_test/A_test.jpg"
img = cv2.imread(image_path)
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
img = cv2.resize(img, (64,64))

# preprocessing the image
img = tf.keras.applications.mobilenet_v2.preprocess_input(img)

# Making predicitions on the image
predictions = model.predict(np.array([img]))

# Get the predicted class label

predicted_class = labels[np.argmax(predictions)]

print(f"The predicted class is {predicted_class}")
```

[37]

```
1/1 [==============================] - 0s 344ms/step
The predicted class is A
```

Test - 2

```
#Load the saved model
model = tf.keras.models.load_model("/content/asl_vgg16_best_weights.h5")
# Load the test image
image_path = "/content/asl-alphabet/asl_alphabet_test/asl_alphabet_test/C_test.jpg"
img = cv2.imread(image_path)
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
img = cv2.resize(img, (64,64))

# preprocessing the image
img = tf.keras.applications.mobilenet_v2.preprocess_input(img)

# Making predicitions on the image
predictions = model.predict(np.array([img]))

# Get the predicted class label

predicted_class = labels[np.argmax(predictions)]

print(f"The predicted class is {predicted_class}")
```

```
1/1 [==============================] - 0s 218ms/step
The predicted class is C
```

Test - 3

```
#Load the saved model
model = tf.keras.models.load_model("/content/asl_vgg16_best_weights.h5")
# Load the test image
image_path = "/content/asl-alphabet/asl_alphabet_test/asl_alphabet_test/nothing_test.jpg"
img = cv2.imread(image_path)
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
img = cv2.resize(img, (64,64))

# preprocessing the image
img = tf.keras.applications.mobilenet_v2.preprocess_input(img)

# Making predicitions on the image
predictions = model.predict(np.array([img]))

# Get the predicted class label

predicted_class = labels[np.argmax(predictions)]

print(f"The predicted class is {predicted_class}")
```

[39]

```
1/1 [==============================] - 0s 221ms/step
The predicted class is nothing
```

Test - 4

```
#Load the saved model
model = tf.keras.models.load_model("/content/asl_vgg16_best_weights.h5")
# Load the test image
image_path = "/content/asl-alphabet/asl_alphabet_test/asl_alphabet_test/space_test.jpg"
img = cv2.imread(image_path)
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
img = cv2.resize(img, (64,64))

# preprocessing the image
img = tf.keras.applications.mobilenet_v2.preprocess_input(img)

# Making predicitions on the image
predictions = model.predict(np.array([img]))

# Get the predicted class label

predicted_class = labels[np.argmax(predictions)]

print(f"The predicted class is {predicted_class}")
```

```
WARNING:tensorflow:5 out of the last 5804 calls to <function Model.make_predict_function.<locals>.predict_function at 0x7fae4ea03370> triggered tf.functior
1/1 [==============================] - 0s 232ms/step
The predicted class is space
```

Text file in which all alphabets are saved

classes.txt

```
1   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 space nothing
```

WSGI Flask Server Code:

Python

```python
from flask import Flask, render_template, Response
from flask_socketio import SocketIO, emit, send
import cv2
import numpy as np
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing.image import ImageDataGenerator

app = Flask(__name__)
socketio = SocketIO(app)

# Data preprocessing for the model
data_generator = ImageDataGenerator(samplewise_center=True,
samplewise_std_normalization=True)

# Loading the ASL model
model = load_model('asl_alphabet_model.h5')

# Setting up image and frame sizes
IMAGE_SIZE = 200
CROP_SIZE = 400

# Loading classes for ASL alphabet
with open("classes.txt") as classes_file:
  classes_string = classes_file.readline()
  classes = sorted(classes_string.split())

# Preparing cv2 for webcam feed
cap = cv2.VideoCapture(0)

# Variables for storing message and last prediction
message = ''
last_prediction = None

# Flask-SocketIO event for handling message emission
@socketio.on('speak')
def handle_speak():
  global message
  send(message)

# Flask route for the home page
@app.route('/')
def index():
  return render_template('home.html')

# Flask route for the demo page
```

```
@app.route('/demo')
def demo():
  return render_template('demo.html')

# Flask route for video feed
@app.route('/video_feed')
def video_feed():
  return Response(gen_frames(), mimetype='multipart/x-mixed-replace;
boundary=frame')

def gen_frames():
  global message, last_prediction
  while True:
    # Capture frame-by-frame
    ret, frame = cap.read()

    # Target area for hand gestures
    cv2.rectangle(frame, (0, 0), (CROP_SIZE, CROP_SIZE), (0, 255, 0), 3)

    # Preprocess the frame before input to the model
    cropped_image = frame[0:CROP_SIZE, 0:CROP_SIZE]
    resized_frame = cv2.resize(cropped_image, (IMAGE_SIZE, IMAGE_SIZE))
    reshaped_frame = np.array(resized_frame).reshape((1, IMAGE_SIZE,
IMAGE_SIZE, 3))
    frame_for_model = data_generator.standardize(np.float64(reshaped_frame))

    # Predict the frame
    prediction = np.array(model.predict(frame_for_model))
    predicted_class = classes[prediction.argmax()]

    frame = cv2.flip(frame, 1)

    # Prepare output based on model's confidence
    prediction_probability = prediction[0, prediction.argmax()]
    if prediction_probability > 0.6:
      # High confidence
      cv2.putText(frame, '          {} - {:.2f}%'.format(predicted_class,
prediction_probability * 100),
            (10, 450), 1, 2, (255, 255, 0), 2, cv2.LINE_AA)
      # Append predicted class to message if different from last prediction
      if predicted_class != last_prediction:
        if predicted_class == 'space':
          message += ' '
        elif predicted_class == 'nothing':
          message += ""
        else:
          message += predicted_class
        last_prediction = predicted_class
    elif 0.2 < prediction_probability <= 0.6:
```

```
      # Low confidence
      cv2.putText(frame, '          Maybe {}... -
{:.2f}%'.format(predicted_class, prediction_probability * 100),
            (10, 450), 1, 2, (0, 255, 255), 2, cv2.LINE_AA)
    else:
      # No confidence
      cv2.putText(frame, classes[-2], (10, 450), 1, 2, (255, 255, 0), 2,
cv2.LINE_AA)

    # Encode the frame as a JPEG image
    _, buffer = cv2.imencode('.jpg', frame)
    frame = buffer.tobytes()

    # Yield the frame as a Flask response
    yield (b'--frame\r\n'
       b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n')

    # Emit the message to the client
    socketio.emit('message', message)

if __name__ == "__main__":
  socketio.run(app, debug=True)
```

Classes.txt File:

```
Unset
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 space nothing
```

Home-Page HTML file:

```
Unset
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>ASL Alphabet Image Recognition</title>
  <link rel="stylesheet" type="text/css" href="{{ url_for('static',
filename='home.css') }}">
```

```
</head>
<body>
  <!-- Home Page -->
  <div id="home">
    <h1>ASL Alphabet Image Recognition</h1>
    <p>
      The American Sign Language (ASL) is the primary language used by deaf individuals in North America. It is a visual language that combines hand gestures, facial expressions, and body movements to convey meaning.
    </p>
    <p>
      In recent years, there has been an increasing interest in developing technologies to help bridge the communication gap between the deaf and hearing communities.
      One such technology is ASL Alphabet Image Recognition, which 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".
    </p>
    <p>
      The trained model can be used to develop applications that can recognize the ASL alphabet from real-time video streams, which could be used to improve communication between the deaf and hearing communities.
    </p>
    <button onclick="location.href='/demo'">Try Demo</button>
  </div>
</body>
</html>
```

Home-Page CSS file:

Unset

```
body {
  font-family: Arial, sans-serif;
  margin: 0;
  padding: 0;
  background-image: url('https://wallpapercave.com/wp/wp4047763.jpg');
  background-size: cover;
  color: #fff;
  backdrop-filter: blur(5px);
```

```
}

h1 {
  font-size: 2.5em;
  padding: 20px;
}

p {
  font-size: 1.5em;
  padding: 20px;
  transition: color 0.3s ease;
}

p:hover {
  color: #f00;
}

#home {
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  height: 100vh;
  text-align: center;
  padding: 20px;
}

button {
  padding: 15px 30px;
  margin-top: 20px;
  border: none;
  border-radius: 50px;
  background-color: #f00;
  color: #fff;
  cursor: pointer;
  transition: background-color 0.3s ease;
  font-size: 1.5em;
}

button:hover {
  background-color: #a00;
}
```

Demo-Page HTML file:

Unset

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>ASL Alphabet Image Recognition</title>
  <link rel="stylesheet" type="text/css" href="{{ url_for('static',
filename='demo.css') }}">
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/socket.io/4.0.1/socket.io.min.js
"></script>
  <script>
    window.onload = function() {
      var socket = io.connect('http://' + document.domain + ':' + location.port);
      socket.on('connect', function() {
        console.log('Connected!');
      });
      socket.on('message', function(msg) {
        document.getElementById('message').innerHTML = msg;
      });

      // Add event listener to the "Speak" button
      document.getElementById('speakButton').addEventListener('click',
function() {
        // Get the message from the paragraph with id 'message'
        var message = document.getElementById('message').innerHTML;

        // Create a new SpeechSynthesisUtterance object with the message
        var utterance = new SpeechSynthesisUtterance(message);

        // Pass the utterance to speechSynthesis.speak()
        window.speechSynthesis.speak(utterance);
      });
    }
  </script>
</head>
<body>
  <div id="videoContainer">
    <img src="{{ url_for('video_feed') }}" id="video">
  </div>
  <div id="outputContainer">
    <p id="message"></p>
    <button id="speakButton">Speak</button>
  </div>
</body>
</html>
```

Demo-Page CSS file:

```
Unset
body {
  font-family: Arial, sans-serif;
  margin: 0;
  padding: 0;
  background-image: url('https://wallpapercave.com/wp/wp4047763.jpg');
  background-size: cover;
  color: #fff;
  backdrop-filter: blur(5px);
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  height: 100vh;
  overflow: hidden;
}

#videoContainer {
  width: 60%;
  height: 60%;
  position: relative;
  margin: 0 auto;
  padding: 20px;
  overflow: hidden;
}

#video {
  position: absolute;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
  object-fit: cover;
}

#outputContainer {
  width: 60%;
  height: 40%;
  border: 1px solid #fff;
  margin: 20px auto;
  padding: 20px;
  overflow: auto;
```

```
  display: flex;
  flex-direction: column;
  justify-content: space-between;
}

#message {
  margin: 0;
}

#speakButton {
  display: block;
  width: 100%;
  height: 50px;
  background-color: #f00;
  color: #fff;
  border: none;
  border-radius: 50px;
  font-size: 1.5em;
  cursor: pointer;
  transition: background-color 0.3s ease;
}

#speakButton:hover {
  background-color: #a00;
}
```

13.2 Git Hub Link:

- GitHub Repo
- Project Demo