Cairo University
Faculty of Computers and Artificial Intelligence
Department of Software Engineering

Sign Language Detection (Talking Signs)

Supervised by:
Dr. Khaled Wassif.
TA. Mohamed Samir.

Implemented by:

Names	IDS	Department
Youssra Mostafa Youssef	20176033	SW
Kareem Ehab Ahmed	20176020	SW
Donia Mahmoud El-Sayed	20176038	SW
Youssef Ahmed Hossam	20175022	SW
Hady Raed Mahmoud	20175019	SW

Graduation Project
Academic Year 2020-2021
Final Documentation

Table of Contents

List of Figures

List of Tables

Chapter 1: Introduction

1- Introduction:

Abstract:

Sign Language is considered one of the most important way of communication among the hearing impaired/mute people and between them and the rest of the society of normal hearing people. The reason for it being the main way of communication is that it depends on visual signs and gestures using hands, arms, facial expressions, and varying body postures to deliver its semantic meaning. Disregarding with that the use of oral communication and sound patterns which require a lot more effort from both parties, whether the hearing-impaired person or the normal hearing one, which makes it an easier way of communication. Now that we talked about the crucial role that sign language play in connecting two parts of the society together, our problem shines here on how we can bring that language to both parties.

Despite sign language being an easy and important way of communication that connects the normal hearing and hearing-impaired people together, it's still facing obstacles in the normal hearing community. That's because of how normal hearing society made no attempt of teaching sign language in schools; and our survey shows how only 45 persons out of 164 (27.4%) is willing to go and learn sign language on their own. And it shows that out of 164 people, there are 117 persons who know nothing about sign language. This resulted in a huge gap in our society as it gets harder to communicate with a certain sector inside it and no attempt is being made to try to minimize this gap. This leaves the deaf and mute community in a kind of a blind spot in the society and almost invisible to the rest of us. Studies also revealed that deaf people are more likely to suffer from psychological problems due to their feeling of loneliness, anxiety, and depression as they often feel isolated and lack the ability to communicate with other people normally. Not only struggling in communication is what they are suffering from, but the lack of communication also negatively impacts their learning process and career choices.

❖ 1.1- Motivation to Solve the Problem:

We were motivated in solving this problem by using deep learning algorithms to help in minimizing the gap between the normal hearing people and the hearing-impaired ones. To be able to give the deaf and mute community a solid ground to start their way of being more active and present in the society. And at the same time requesting no learning effort from neither the normal hearing people nor the hearing-impaired, so that they can be encouraged to use the application with no second thoughts or doubts. All that our system does is that it captures the signs made by hand gestures and translates them to letters so that it can form words in text and audio, relying on the American Sign Language as the language being used by the hearing impaired person.

Our model considers the similarities of human hand shape with four fingers and one thumb and aims to present a real time recognition of the hand gestures being signed depending on the detection of some shape based features like orientation, centre of mass centroid, fingers status, thumb in positions of raised or folded fingers of hand.

The core of our project was to develop a CNN model to detect signs with high accuracy and real time performance, relying on the transfer learning approach to achieve this, thereby making a research contribution in this field. However, we thought about how it will be more useful if we made this model available to be used by the end user, so that he can truly benefit from it, and by that we will not only be making a research contribution, but also delivering

the value of this contribution for the customer to use. We decided that we can deliver the value of the model by making it a web-service; meaning that we will host the trained model on a server with declared port for it and a known API link that any application or website can send a post request through it to the model and get a response which will be the translation of the image sent in the request. By doing this, we would have gained the most value from this model and made it available for any system to use easily as it became a web-service. In addition to this, we wanted to demonstrate how the communication with the model as a web-service would be like, so we developed a simple android mobile application using React Native, that a deaf-and-mute person can use, by capturing an image of the performed sign and communicating with the model through sending a post request of the captured frame to the server, and receiving a response from the server with the translated letter to show on the screen for the user to see, and giving the user the choice to translate the text shown on the screen to audio as well.

So the hearing impaired user that's using the application will do nothing more than opening the app and start signing his letters, as for the normal hearing user he can then read or listen to the translation, making the application easy to use and non-profitable for both users.

❖ 1.2- Problem Definition:

Hearing impaired and mute people have difficulty in communicating with normal people as their sign language is not understood by them. So, implementing a CNN model that can detect American Sign Language and translate these signs into text, then hosting the trained model on a server making it a web-service, gives other systems and end-users the opportunity to communicate with the model easily and exposing its benefits to the hearing-impaired community, thus reducing the gap in our society. Also, demonstrating how the communication with the model as a web-service can take place by developing an android mobile application that does so, thereby delivering the outmost value for the user.

Research Question:

The research question we are trying to answer here is: Can we provide the normal hearing and hearing-impaired communities a way to communicate that is easy, non-profitable, accurate, and requires no learning effort from both sides?

❖ 1.3- Project Objective:

The main area of our project is developing a CNN model using deep learning techniques and algorithms to detect signs and translate it to text and audio. Aiming to improve the accuracy of the model using optimization techniques to give better results than other researches made about this topic. Hosting the previous model on a server to transform it into a web-service, so that it can be really used by the deaf and mute community and not leaving it as just a model with a better accuracy that cannot provide an impactful value to the community. Showing off the value of this web-service by developing a cross platform application using the web-service and providing the user with the ability to open the application and start performing signs to be translated in text or speech in real-time and by that we would facilitate the communication with the deaf and mute community.

❖ 1.4- The Gantt Chart and work plan of the Project:

➢ 1.4.1- Work Plan:

Task ID	Task Title	Task Description	Task Status
1	Search for relevant research papers.	-This task is concerned with gathering and reading research papers that are related to the same idea we are working on to gather enough knowledge. -This task is the responsibility of every team member.	Completed in 3 weeks.
2	Study Machine Learning Algorithms.	-This task is concerned with learning machine learning from scratch to understand more about the nature of the model we are developing. -This task is the responsibility of every team member.	Done. (Started from 10/11). (Ended on 15th of July).
3	Study Deep Learning Techniques.	-This task is concerned with learning more about deep learning to understand more about the nature of the model we are implementing. -This task is the responsibility of every team member.	Done. (Started from 10/11). (Ended on 15th of July).
4	Read and understand CNN.	-This task is concerned with knowing how CNN works and the main differences between it and ANN. -This task is the responsibility of every team member.	This is a recurring task that we keep going back to; so that we can make sure we are going the right way. (Completed on the day of submission: 26/7)
5	Decide on the main features. (Milestone).	-We needed to explicitly define the main features that will be provided by our idea. Improving accuracy and providing the end-user a real value. -This task is the responsibility of every team member.	Completed in 5 days.

Task ID	Task Title	Task Description	Task Status
6	Take a React Native online course.	-We needed to know how to develop mobile application using the react native framework. -This task is assigned to Youssef.	Completed in 2 months.
7	Search and understand the optimization techniques.	-This task is concerned with knowing more about optimization techniques that can be applied on models to improve its training and prediction. -This task is the responsibility of Youssra, Donia & Kareem.	A recurring task like all the tasks that involves studying and knowledge.
8	Choose techniques to be used in the model.	-This task is concerned with settling on the final techniques that will be used in the model. -This task is the responsibility of every team member.	Completed in 3 days.
9	Practicing and understanding certain tools.	-This task is concerned with learning more about TensorFlow, OpenCV and NumPy. -This task is the responsibility of every team member.	Completed in 3 months. Finished on 15th of May.
10	Conduct a survey and gather the responses.	-This task is for collecting statistics, opinions, and concerns about our idea. -This task is assigned to Hady.	Completed in 2 weeks.
11	Data Acquisition.	-This task is reserved for searching for appropriate experimental datasets that can be used in our model. -This task is the responsibility of Youssra & Donia.	Completed in 5 days.
12	Designing application's logo.	-This task is for creating a unique logo that defines the application we are developing. -This task is assigned to Hady.	Completed in 2 days.

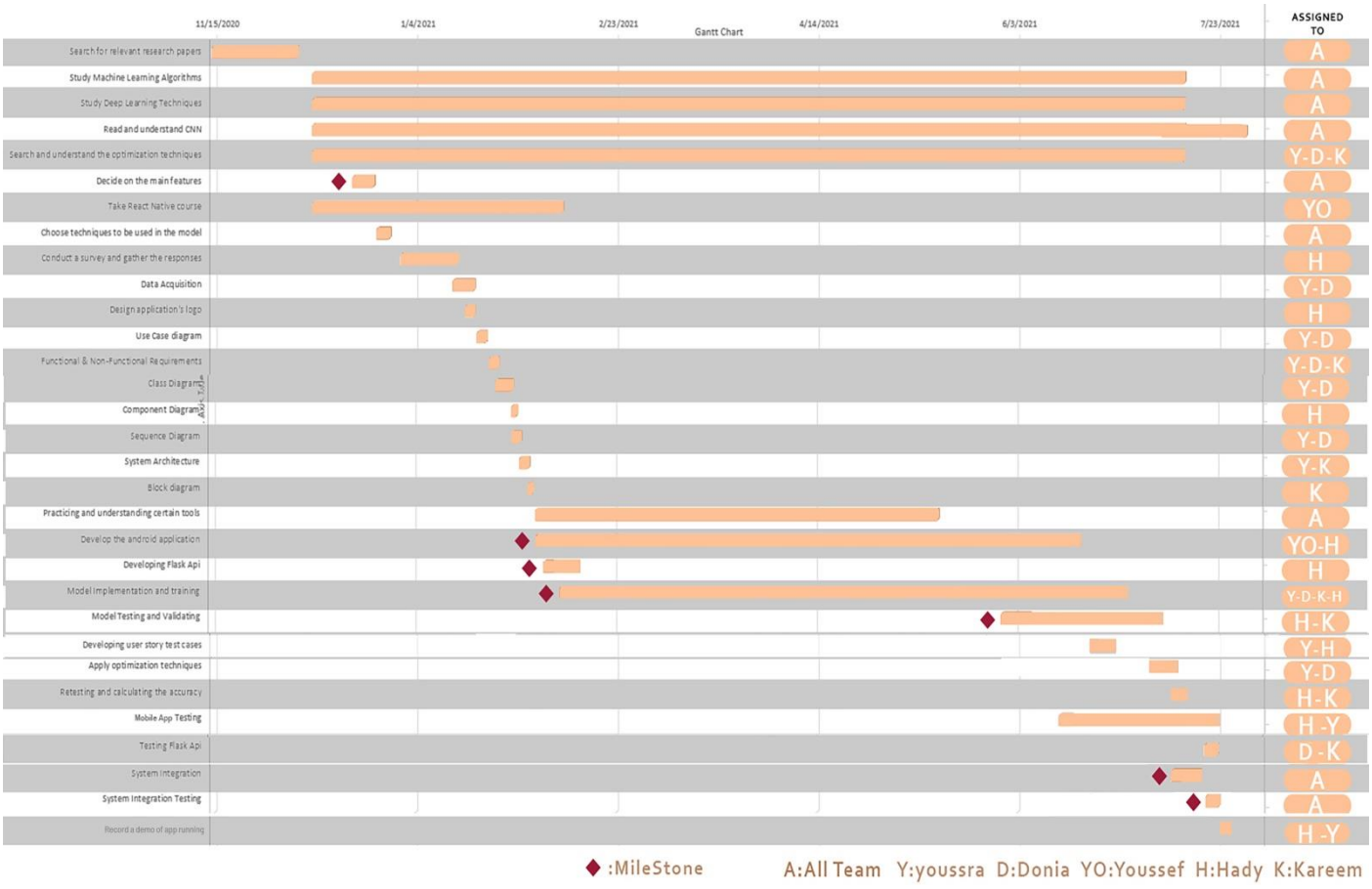
Task ID	Task Title	Task Description	Task Status
13	Use Case Diagram	-This task is for creating the use case diagram and each use case sample within it. -This task is the responsibility of Youssra & Donia.	Completed in 2 days.
14	Functional & Non-Functional Requirements.	-To list the functional & non-functional requirements of our app. -This task is the responsibility of Youssra, Donia & Kareem.	Completed in 2 days.
15	Developing user story test cases.	-To document test cases that represent scenarios on the use case diagram to validate the user stories. -This task is assigned to Youssra & Hady.	Done. Started on the 10/7. Finished on 12/7.
16	Class Diagram.	-To create the detailed class diagram that reflects the internal structure of the system. -This task is the responsibility of Youssra & Donia.	Completed in 4 days.
17	Component Diagram	-To identify the components of the system and the classes grouped in each component and help in designing the system architecture. -This task is assigned to Hady.	Completed in 1 day.
18	Sequence Diagram.	-To create a sequence diagram that shows the interaction between the system components. - This task is the responsibility of Youssra & Donia.	Completed in 2 days.
19	System Architecture.	-To decide and design the suitable architecture for the system. - This task is the responsibility of Youssra & Kareem.	Completed in 3 days.

Task ID	Task Title	Task Description	Task Status
20	Develop the android mobile application. (Milestone).	-To start developing the application using react native and capturing real time video from the end user and displaying the translation on screen. -This task is assigned to Youssef & Hady.	Done. Expected to take 45 days. But took 2 months. Ended on 9/7.
21	Develop the Flask API. (Milestone).	-Developing an API using Flask framework, to as act as an intermediary between the application and the model to pass frames and text to and from. -This task is assigned to Hady.	Completed in 5 days.
22	Block Diagram.	-Create block diagram to show the flow of our system. -This task is assigned to Karim.	Completed in one day.
23	Model Implementation and training. (Milestone).	-To develop the CNN model, apply the transfer learning technique and train it on the selected dataset. -This task is assigned to Youssra, Donia, Kareem and Hady.	Completed on 14/7. Took one month instead of 4 months.
24	Model Testing and Validating. (Milestone).	-To test the model on unlabelled dataset and calculate the accuracy. -This task is assigned to Kareem and Hady.	Completed in 2 days. Started after training is done. Finished on 16/7.
25	Apply optimization techniques.	-To decide on and apply the suitable optimization technique to enhance and improve the calculated accuracy. -This task is assigned to Youssra & Donia.	Completed in 2 days after testing was done. Finished on 18/7.

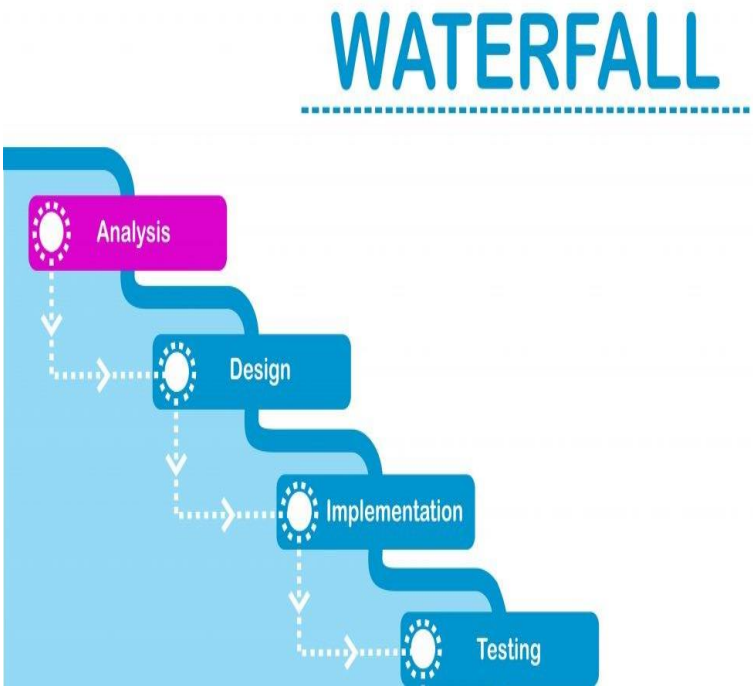
Task ID	Task Title	Task Description	Task Status
26	Retesting and calculating the accuracy.	-To retest the model after the optimization technique is applied and re-calculates the improved accuracy. -This task is assigned to Kareem and Hady.	Completed in 1 day after applying optimization technique was done. Finished on 19/7.
27	Integrating the 3 main modules. (Milestone).	-To integrate the mobile application with the Flask API, and the Flask API to the classifier model to pass the frames and interpreted letters. -This task is the responsibility of all team members.	Completed in 2 days. Finished on 21/7.
28	Mobile Application Testing. (Testing Frontend).	-To test the frontend of the mobile application. (Based on documented test cases). - This task is assigned to Youssra and Hady.	Completed in 2 days. Finished on 23/7.
29	Testing the Flask API and the web-service.	- This task is assigned to Kareem & Donia.	Completed in 2 days. Finished on 23/7.
30	System Integration Testing. (Milestone).	-To test that the 3 components work together with no failures or defects. -This task is the responsibility of all team members.	Completed in 2 days. Finished on 25/7.
31	Recording a demo of the application running.	-To record a video of one of us using the application as a demo. -This task is the responsibility of Youssef & Hady.	Completed in 1 day. Finished on 26/7.

Table 1.1

➢ 1.4.2- Gantt Chart:



❖ 1.5- Project Development Methodology:



- The development methodology followed in this project is the Waterfall methodology model, which reflects the approach we took in tackling this project in a linear sequential way as the waterfall shows.

- We started first with the Requirements Analysis Phase, this included eliciting of functional and non-functional requirements and trying to gather information about the machine learning and deep learning techniques that can be used to achieve our model. This phase also included the survey we conducted to gain knowledge about whether the idea behind our project will be of value to people or not, and how to carry it out in the most useful way for the user. Not to mention, the research papers we went through in this phase that addressed the same area of the model we wanted to implement.

- Next phase we went through was the System Design Phase, in this phase we started to select the system architecture that our project will have. We demonstrated the use cases in our system with a use case diagram and user stories, defining by that the stakeholders and the actors that will participate somehow in our project. Following that we designed a class diagram and a component diagram to show the structural view of the system using objects, attributes, operations and relationships that exist in the problem domain. We then represented the dynamic interactions that take place during runtime between the system objects and components and the overall behaviour of the system, showing collaborations among objects and changes to the internal states of objects by designing sequence diagram for each use case scenario and a block diagram for the proposed system.

- Then follows the Implementation Phase that we spent developing systems' parts in parallel. First, three of us started developing the CNN model and training it on the chosen dataset, repeating the training process while changing some factors like the optimizer used or the number of epochs to train so that the model can reach the best validation accuracy it could without over-fitting. Meanwhile, the other two team members were developing the android mobile application using React Native, so that when the model is ready they can start hosting the trained model on the Digital Ocean Server, thereby creating the web-service that the mobile application can communicate with through a post request.

- After the Implementation Phase was finished, we went on to perform testing. The Testing Phase was divided into three parts, two of which were done in parallel. First, the frontend of the mobile application was tested and in parallel, the remaining team were creating a testing dataset consisting of 125 images to be used in a python script that simulate the communication with the web-service by sending each image as a post request to the server, then checking if the response matches the actual class of the sent image or not, and calculating the testing accuracy using the hit-miss technique. The final part of testing was system testing that was done to test different scenarios on the system as a whole; this included functional and non-functional testing, to simulate the communication among all the modules in our system.

- Our final phase was solving the defects found in the previous Testing Phase to make sure that the system comes out in the best possible way we could. Making improvements to the model and its accuracy was also included in this phase.

❖ 1.6- The Used Tools in the project (SW and HW):

- Programming Language used to build the model is: Python v3.7.4 or higher (should work with v3.5.2 and above as well).

1.6.1 Python

- The mobile app is built using React Native Framework.

1.6.2 React Native Framework

- TensorFlow software library is used in developing and training our model and to easily deploy it on a device. Required version is v1.15.0-rc3 (may work on higher versions) [GPU version preferred].

1.6.3 TensorFlow

- NumPy is a Python library that deals with multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays which we needed to work on the feature vectors and matrices that our CNN works on.

1.6.4 NumPy

- OpenCV v3 is required. It is used to capture real time frames from a video feed and analyse the features in each frame.

1.6.5 OpenCV

- Flask API is used as an intermediary between the mobile application and the model; passing each frame received from the mobile application as a post request to the model to be detected.

1.6.6 Flask

- Google Colab (NVIDIA Tesla K80 GPU) is used for training and testing the model on browser with faster Graphics Processing Unit (GPU).

1.6.7 Google Colab

- GitHub is used as our code repository; so that the team can contribute to the code simultaneously and as a version control and branching tool.
  (GitHub Link: https://github.com/YoussefAhmed99/talking-signs)

1.6.8 GitHub

- DigitalOcean was used as cloud infrastructure provider that provides cloud computing services for developers, such as hosting their applications and systems on their cloud and thereby providing high CPU power and availability of the system. Not to mention, the multithreading and concurrency it provides for accessing the hosted application.

1.6.9 DigitalOcean

- NGINX is open-source software for web-serving, provided by DigitalOcean. This is used to route requests coming from other systems and applications to our hosted model securely using Secure Socket Layer. Thus making the HTTP post request, a (secured) HTTPS post request.

1.6.10 Nginx

- Docker was used to apply the containerisation concept. This was done by placing the model hosted on DigitalOcean in a container with all its configuration files and needed library packages. Containers are separated from one another and this gives us better modifiability, as each functionality or feature will be isolated in a container of their own, without changing anything in the already existing container. And communication between the containers and each other is done through channels easily.

1.6.11 Docker

❖ 1.7- Report Organization:

➢ Our report is organized in the following manner:

- In Chapter 2, we go through the project's related work and similar implementations of it, emphasizing the main differences between these implementations and ours, and showing the advantages and disadvantages of each one.
- In Chapter 3, we dive into the system analysis of the project. This chapter discusses the functional and non-functional requirements with the important stakeholders of the project. Also, it elicits the use cases we have and represents them in a use case diagram.
- In Chapter 4, the research problem starts to shine and gets a deep explanation about the research question we are trying to answer, the dataset we used, the format of the model's inputs and outputs. Not to mention, going through the different obstacles we faced during the project; whether they are technical, computational or financial obstacles.
- In Chapter 5, here we take a deep look at how the system is designed through a group of structural diagrams that represent the objects, components and operations that exist in the system and the relationships between them; such as class diagram and component diagram. And behavioural diagrams that represent the dynamic interactions that occur between the system's objects and components during runtime; such as block diagram and sequence diagram. Also, the architecture we chose for the system is represented through the system architecture diagram and a justification for our choice is given. Not to mention that this chapter shows through screenshots the prototype and the system GUI design, showing the differences between the initial prototype and the final real design we implemented.
- In Chapter 6, we go through the implementation and testing process in great detail, discussing all techniques used, approaches followed and frameworks or tools used, and the advantage and limitation of each and everything. There are code snippets provided for some of the implementation parts to visualize some of the explanation. The testing process is also fully explained with test cases for every type of testing that was carried out, including the testing code. And provided in this chapter is the GitHub Link that's the repository of our project, containing the whole project code.
- Finally, we have the references part referencing all research papers and any citations mentioned in the document with the author name mentioned.

❖ 1.8- Gathering Information about The Project:

The survey we conducted can be found at this link:
https://docs.google.com/forms/d/e/1FAIpQLSccNfcLFfp2-0sChcKu7dcUtwOp6pCxr2rrvlzqUJ7mBEu1vA/viewform

We conducted this survey on Facebook among a group of friends and strangers to gather information about the idea of our project, see if it matches the people's needs and how it can impact their lives. The survey results can be represented through the following statistics:

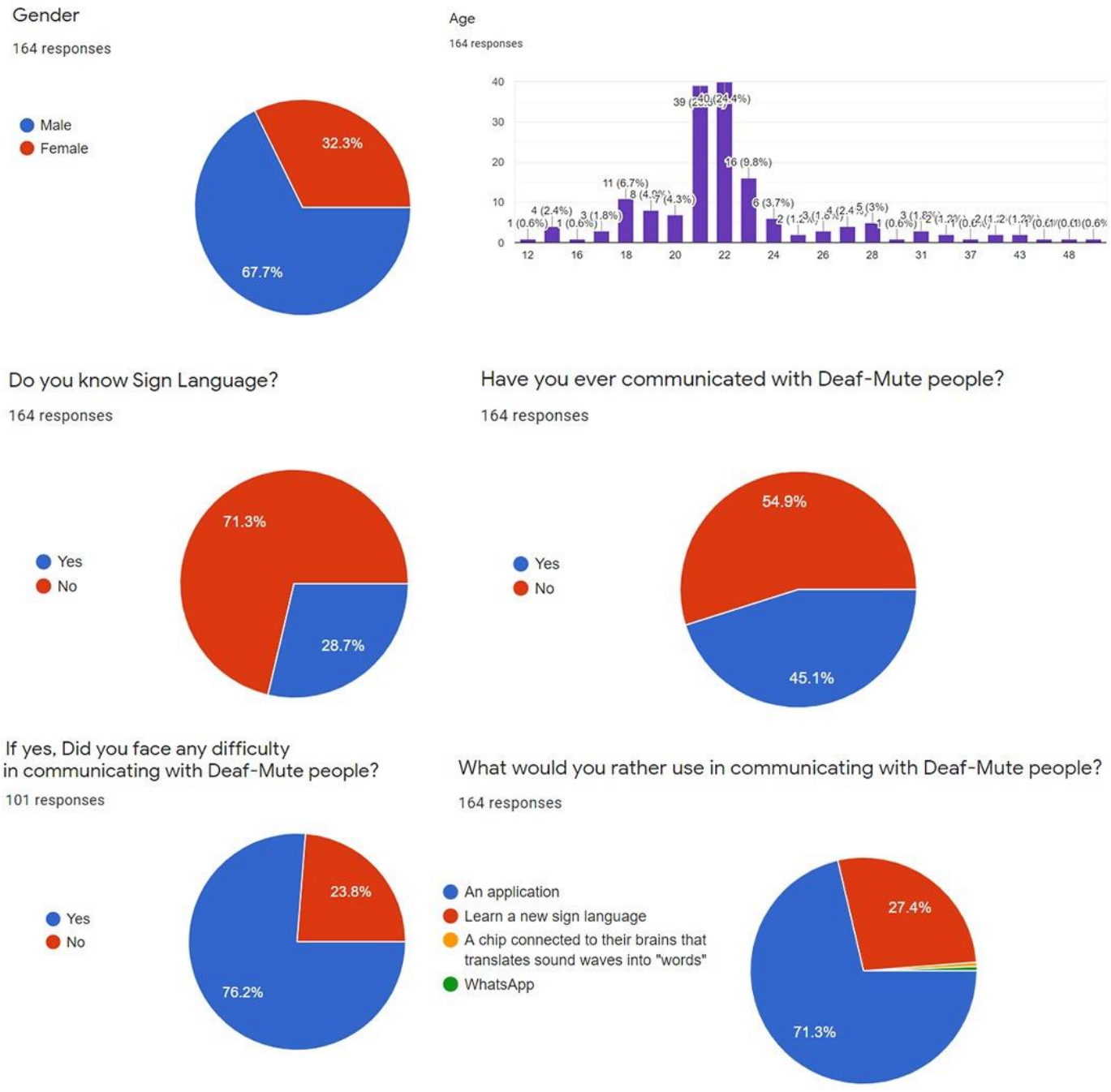


Figure 1.1

Chapter 2:
Related Word

• 2-Related Work:

❖ 2.1- Related Work Concerning The Mobile Application:

➔ **The existing similar implementations of the idea of our project is:**

1- A mobile application called GnoSys:
https://newzhook.com/story/20387/. It detects Sign Language and translates it into text and speech using neural networks and computer vision to do so.
It came out in Netherlands so far and can be used on various devices and tablets.

2- Another app which is similar to our project is
https://ieeexplore.ieee.org/document/7519386, a mobile application of American Sign Language translation via image processing algorithms.
This app is using Speeded up Robust Features (SURF) algorithm and Support Vector Machine (SVM) to classify the feature set. The Classification is done on 16 classes of the ASL and with an overall accuracy of 97.13%.

➔ **The main differences between this app and our project is:**

1- Our application is non profitable and charging no fees to be downloaded and used, unlike GnoSys, it charges fees to use the application.
Moreover, the application translates Indian sign language while we detect American Sign Language, which is much more used worldwide.

2- Our model depends on Transfer Learning technique and has 29 classes that cover all 26 classes of the ASL and not just experimenting on 16 classes of them. Also having three extra classes that represent Space, Delete and None.
Our model aims by using Transfer Learning alongside optimization technique to achieve better accuracy

❖ 2.2- Research papers related to our research point:

Analysis							
Author Name	**Year**	**Name of techniques or algorithms used**	**Accuracy**	**Segmentation**	**Dataset**	**Scope**	**Calculating Accuracy Technique**
[1] Dardas et al.	2010	-Support vector machine (SVM) -k-means clustering - BoF	90%	-	4x100 samples	Gestures	-Average Recognition Rate = (No. of Correct Response) / (No. of Total Samples) * 100 %

Author Name	Year	Name of techniques or algorithms used	Accuracy	Segmentation	Dataset	Scope	Calculating Accuracy Technique
[2] Regina Lionnie, Ivanna K. Timotius & Iwan Setyawan	2012	-K-nearest neighbor (k-NN)(k=1)	83.15%	Edge detection, low pass filtering, histogram equalization, thresholding and desaturation	6x480 samples	Gestures	-
[3] Zaki and Mahmoud	2011	-Hidden Markov Models (HMMS)	89.1%	Skin colour Connected component labelling	RWTH-BOSTON-50 database	ASL	-
[4]Taskiran, M., Killioglu, M., & Kahraman, N	2018	-CNN+ Skin Detection -Convex Hull	-Real time System Accuracy: 98.05% -Test accuracy :100%	-	-Collected in 2011 by Massey University, Institute of Information and Mathematical Sciences -25 samples for each of 36 characters(26 letters and 10 numbers) in the dataset -900 images	ASL	-Average Recognition Rate=(No. of Correct Response) / (No. of Total Samples) * 100 % -360 Tests(5 faults)
[5] Cao Dong, Ming C. Leu and Zhaozheng Yin	2015	Random Forest and Joint Angles with Constraints	-70% (l-o-o) -90% (h-h) - Mean accuracy of 92%	Per-pixel classification algorithm used to segment a human hand into parts	-3,000 images -Publicly available dataset from Surrey University.	ASL	“half-half” (h-h) and “leave one out”(l-o-o) experimental tests.
[6]Md. Mohiminul Islam, Sarah Siddiqua, and Jawata Afnan	2017	-K Convex Hull -ANN	94.32%	Pixel Segmentation	1850 images 50 samples	ASL	-Average Recognition Rate = (No. of Correct Response)/ (No. of Total Samples)* 100% 3700Tests(290 faults)

Author Name	Year	Name of techniques or algorithms used	Accuracy	Segmentation	Dataset	Scope	Calculating Accuracy Technique
[7] Tse-Yu Pan, Li-Yun Lo, Chung-Wei Yeh, Jhe-Wei Li, Hou-Tim Liu, Min-Chun Hu	2016	-SVM -SIFT, Hu-moments and FD PCA and LDA	99.8%(CSL) 94% (ASL)	Skin colour (YCbCr) with Gaussian Mixture Model(GMM)	-26 × 300 samples,7800 Images(CSL dataset collected by themselves) -36 × 2425 samples,2515 Images(public ASL dataset)	-CSL -ASL	-Average Recognition Rate = (No. of Correct Response) / (No. of Total Samples) * 100 %
[8]Gupta et al.	2016	-k-NN -HOG - SIFT	-**HOG** (100% single-handed)(82.77% double-handed) -**SIFT** (92.50% single-handed)(75.55% double-handed) -**Fused features** (97.50% single-handed) (91.11% double-handed)	-	26 × 30 samples	-ISL -Single Handed and Double Handed	-Average Recognition Rate = (No. of Correct Response) / (No. of Total Samples) * 100 % -60 Tests(1 fault for single handed) -200 Tests(21 faults for double handed)

Table 2.2

Our Work				
Name of techniques or algorithms used	**Accuracy**	**Segmentation**	**Dataset**	**Scope**
-Transfer Learning -Data Augmentation -Optimization technique (Adam optimizer).	-Calculated using Hit-Miss Technique. -Training Accuracy: 98% -Validation Accuracy: 91% -Testing Accuracy: 96%-95.2%	Type of segmentation is Similarity Detection following the Region Approach using the CNN technique which provides ready-made libraries.	- 29 Signs(87,000 Images, each image 200x200 pixels) - Testing dataset (125 images) to test the model on.	ASL

Table 2.1

Chapter 3:
System Analysis

❖ 3.1- Project Specification:

The core of our project is a research problem that's aiming to implement a sign language detection model with a combination of techniques that can provide higher accuracy. And also abides by the Application development specifications as we were aiming to deliver the value of sign language detection to the end-user by hosting the trained model on a server, thus creating a web-service that can be used and accessed by any other system or application; so that it can be used by the community. And developing an android mobile application that demonstrate its use of the web-service.

➢ 3.1.1- Functional Requirements:

- ✓ Deaf and mute user should be able to perform a sign that corresponds to an alphabet character in front of the mobile camera.
- ✓ The system should show the translated character of the sign performed in the text area for the user to see.
- ✓ The system should concatenate characters shown in the text area as the user keeps signing.
- ✓ The system should stop the concatenation and leave a space when a space sign is performed by the user.
- ✓ The system should delete the last character shown in the text area when a delete sign is performed or when the delete button is pressed by the user.
- ✓ The user should be able to clear the text area showing the translation by clicking the refresh button.
- ✓ The user should be able to see a manual of all the signs that can be performed when clicking on the Manual button.
- ✓ User should be able to choose the speech translation option for the character/word/sentence shown in the text area.

➢ 3.1.2- Non-Functional Requirements:

- ✓ The camera must be shut off once the user exits the application. (Security).
- ✓ The response time of the application to show the translation for the signed character must not exceed (1) second. (Performance).
- ✓ The user must be asked for camera access when opening the application and the application will be granted access accordingly. (Security).
- ✓ The translation text shown on the screen must be in a 13-pts font size, so that it can be readable for all users. (Usability).
- ✓ The user must be provided access to a manual showing all signs available to help in using the application by performing the right signs. (Usability).

➢ 3.1.3- Stakeholders:

- ✓ Deaf & Mute users.
- ✓ Normal hearing users.

❖ 3.2- Use Case Diagrams:

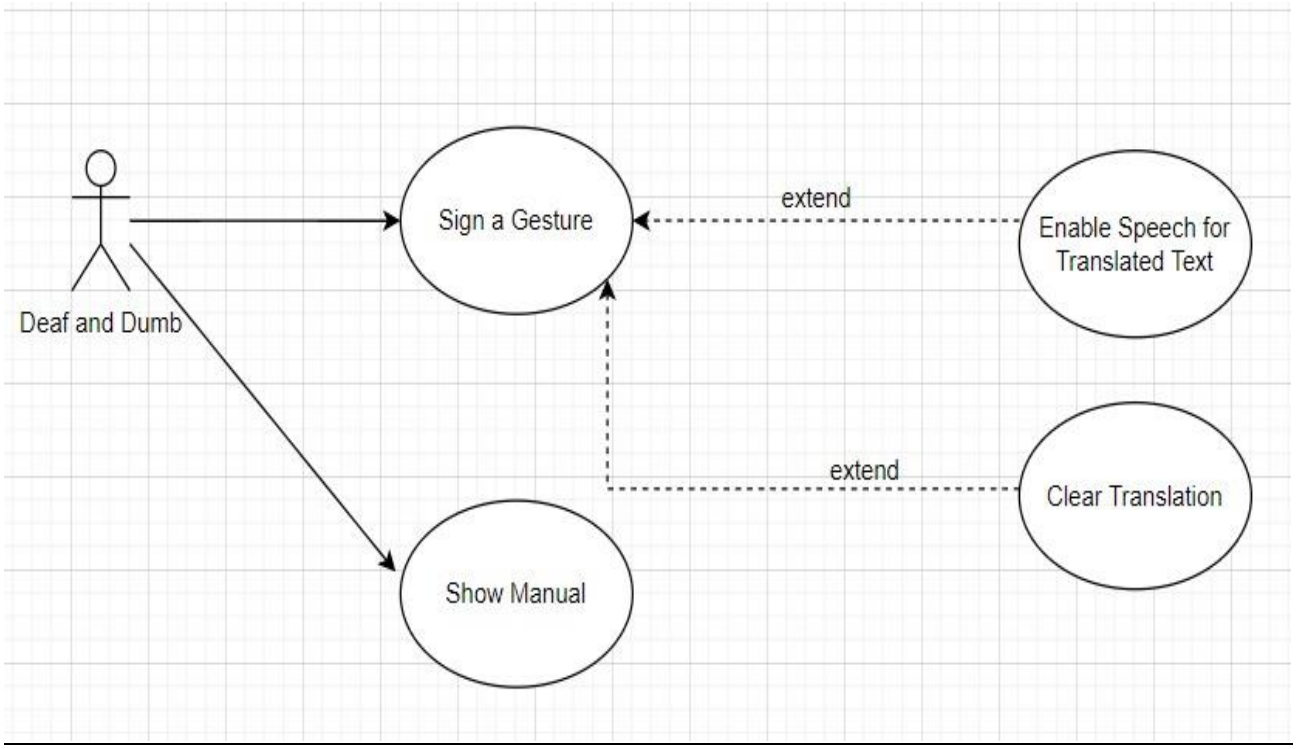


Figure 3.1

Sample Use-Cases:

Overview	
Use Case ID	UC_ID1
Use Case Name	Sign a Gesture.
Description	This use case allows the user to sign a gesture to be translated into a text or an optional audio.
Actors	Deaf and Mute.
Preconditions	Not Applicable.
Basic Flow	
1. The user signs a gesture. 2. The system detects the entered gesture. 3. The system displays the translation. 4. Repeats step 1 to step 3 for all signs. 5. (Extension point: Enable Speech for Translated Text). 6. (Extension point: Clear Translation). 7. This use case ends.	
Post Condition	The application displays the translation of the entered sign to text.
Alternative Flow	None.

Table 3.1

Overview	
Use Case ID	ID_UC2
Use Case Name	Enable Speech for Translated Text.
Description	This use case allows the user to enable an audio option for the translated text.
Actors	Deaf and Mute.
Preconditions	The user signs a gesture to be translated to text and to enable an optional use of audio.
Extend Use Case	Sign a Gesture.
Basic Flow	
1. The user selects the audio option. 2. The system responds with an audio translation for the translated text. 3. This use case ends.	
Post Condition	The system responds with an audio for the text of the translated sign.
Alternative Flow	None.

Table 3.2

Overview	
Use Case ID	ID_UC3
Use Case Name	Clear Translation.
Description	This use case clears the translation field.
Actors	Deaf and Mute.
Preconditions	The user signs a gesture to be translated to text and to enable clearing it.
Extend Use Case	Sign a Gesture.
Basic Flow	
1. The user selects the clear translation option. 2. The system clears the translation field. 3. This use case ends.	
Post Condition	The system clears the translation field.
Alternative Flow	None.

Table 3.3

Overview	
Use Case ID	ID_UC4
Use Case Name	Show Manual.
Description	This use case shows the user a guide for the signs that can be used for translation.
Actors	Deaf and Mute.
Preconditions	None.
Basic Flow	
1. The user selects the show manual option. 2. The system responds with a guide for the signs to be used in translation. 3. This use case ends.	
Post Condition	The system shows a guide containing the signs to be used.
Alternative Flow	None.

Table 3.4

Chapter 4:
Research Problem

❖ 4.1- The Experimental Dataset we are attempting:

- The dataset we are experimenting with our model for training and validation is https://www.kaggle.com/grassknoted/asl-alphabet. This American Sign Language dataset consists of 29 classes; 26 classes corresponding to the 26 English alphabetical letters and the 3 other classes are for SPACE, DELETE and NONE signs. The total number of images in the dataset is 87,000 images; each image is 200x200 pixels. This dataset is augmented as mentioned above using brightness shift (ranging in 20% darker lighting conditions) and zoom shift (zooming out up to 120%), that's to allow classification of images that weren't noise filtered and enhance accuracy.
- Another dataset is used for testing the model after training is done. This dataset contains 125 images, and was created by us during the testing phase of the project. The dataset can be found in the testing chapter with every image. Its actual class and its predicted one.

❖ 4.2- The Format of the system inputs and outputs:

The **inputs** to our application are frames captured by the mobile camera, and sent to the model to be classified. (RGB Frames).

The **output** of our system when frame is passed to the model is a character printed on the translation text area screen. There's an option to get an audio output for the shown text. (char & audio).

❖ 4.3- The Technical/Financial/Computational obstacles we are facing and how we are planning to overcome them:

➢ 4.3.1- Technical Obstacles:

It was very challenging deciding on this idea as it required a deep knowledge of Machine Learning, Deep Learning and CNN which we had few to zero background of. We knew the basics of neural networks but never heard of convolutional before working on this project. We didn't know the mere basics of Machine Learning let alone Deep Learning, and never used tools like TensorFlow or OpenCV.

We are still trying to overcome this obstacle of having minimum knowledge of the tools and techniques we are using by dedicating a heavy schedule along the first semester to study Machine Learning Algorithms and practice with the tools on small projects. We are still learning alongside the implementation but it's proving to be very time consuming. This obstacle was lessened when we took the machine learning course during the second semester.

➢ 4.3.2- Financial Obstacles:

The Google Cloud Platform we aimed to use for training our model is charging a not so affordable amount of money to use it, and the free trial will not be enough to finish the training.

To overcome this we decided to convert to Google Colab for the training of our model which is free.

The DigitalOcean cloud hosting provider we used to host our model and make it a web-service, charged 100$ to host the model for two months, which was a great amount of money that we needed to afford as we didn't have any other options.

➢ 4.3.3- Computational Obstacles:

The Google Colab has some drawbacks to it that we weren't facing in Google Cloud Platform. It only trains for 8 hours and then stops, so training takes more time than we planned. The GPU power offered by Google Colab is actually satisfactory. However, GPU in Google Cloud Platform was much higher and faster. We have nothing to do about this obstacle as we don't have any other options.

Chapter 5:
System Design

❖ 5.1- System Architecture:

The architecture chosen for our system is Layered architecture, as it simply represents the structure of our components of having the Application as the component in the presentation layer that is responsible for displaying UI to the end user. As for the business logic layer we have the Flask API component that acts as an intermediary between the application and the model. The Service layer has one component which is the classifier model that's responsible for predicting the passed frames and thus providing the service.

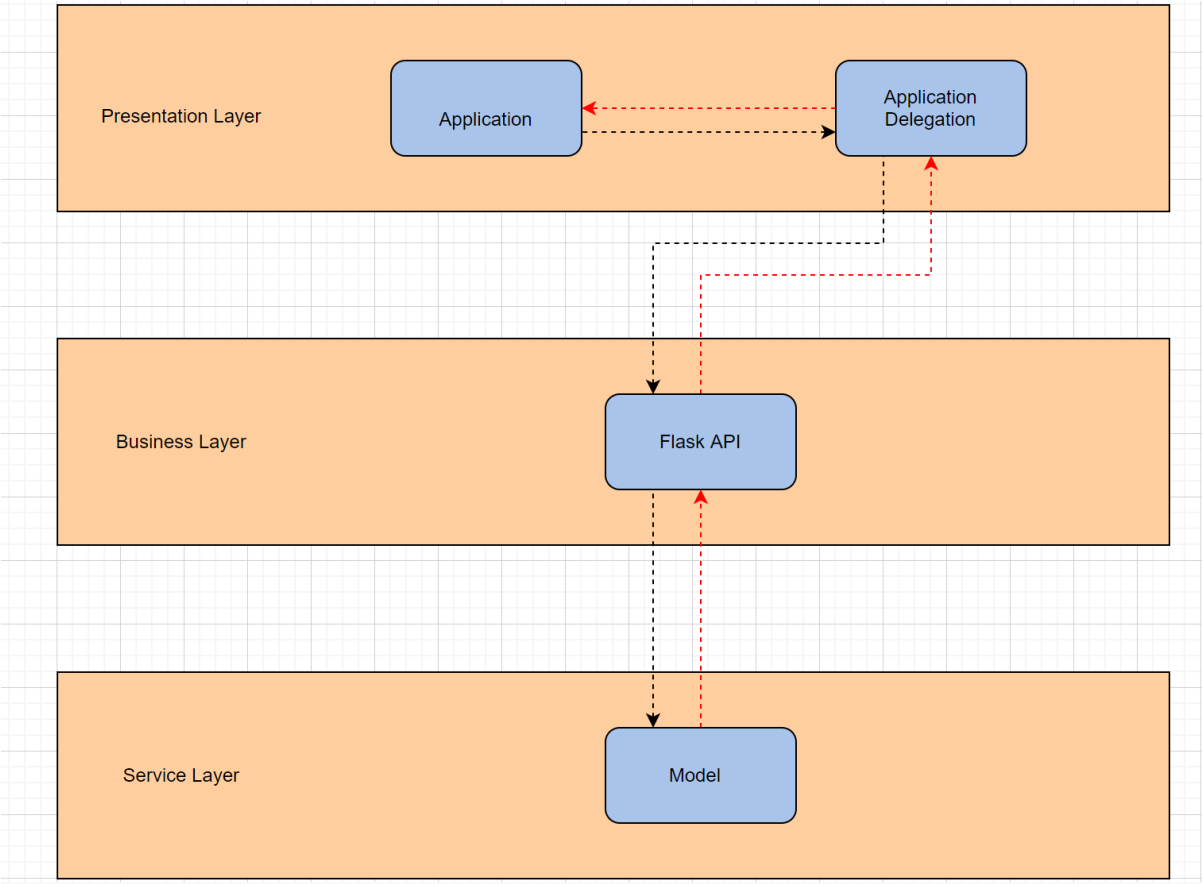


Figure 5.1

❖ 5.2- System Component Diagram:

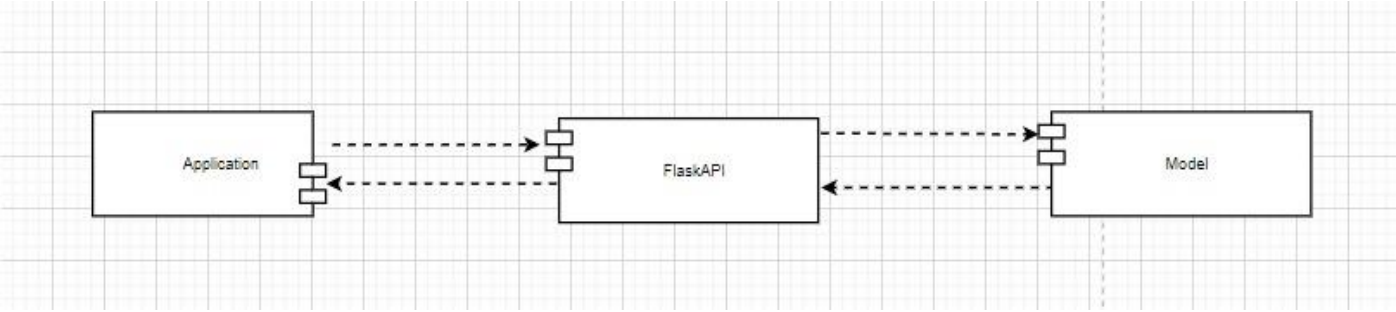


Figure 5.2

❖ 5.3- System Class Diagram:

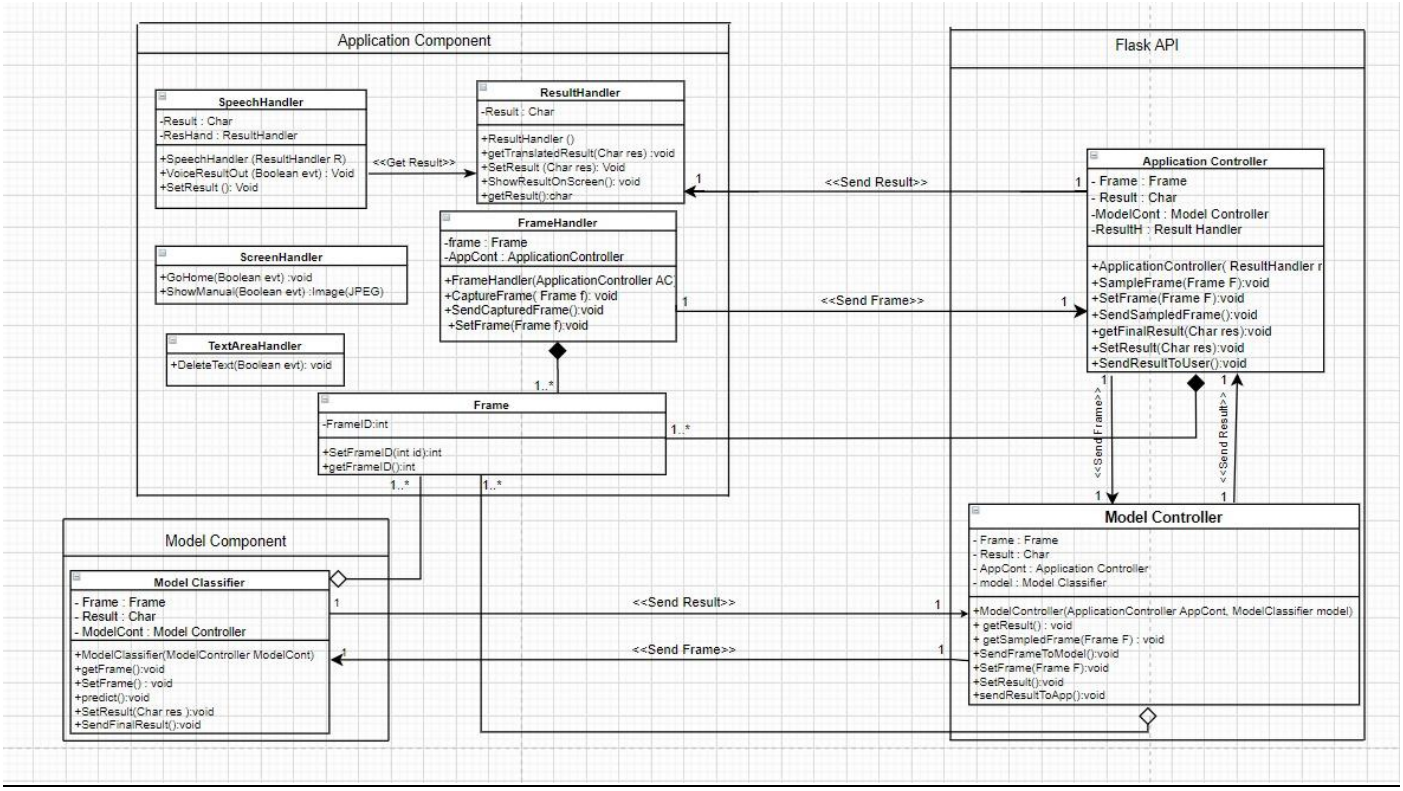


Figure 5.3

❖ 5.4- Sequence Diagram:

Sequence Diagram for Sign a Gesture use-case:

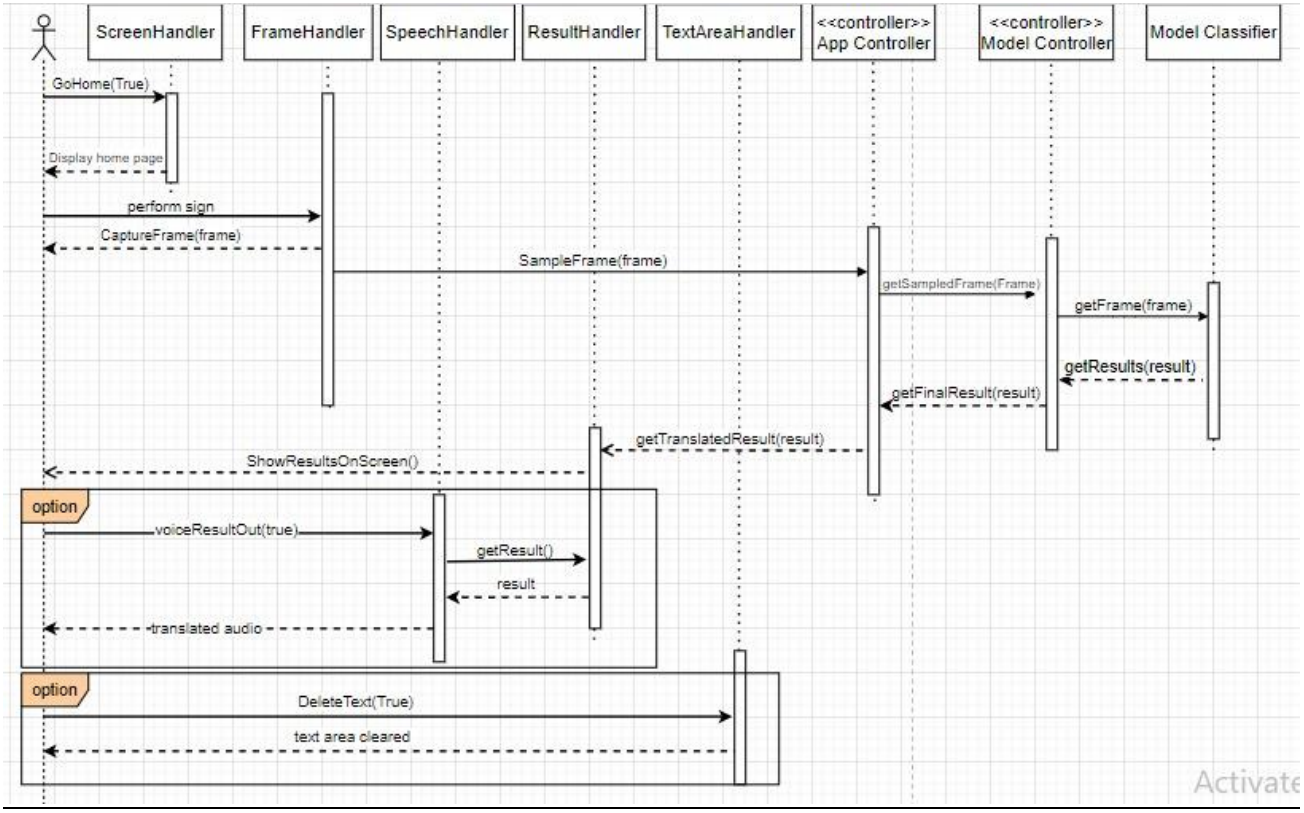


Figure 5.4

Sequence Diagram for Show Manual use-case:

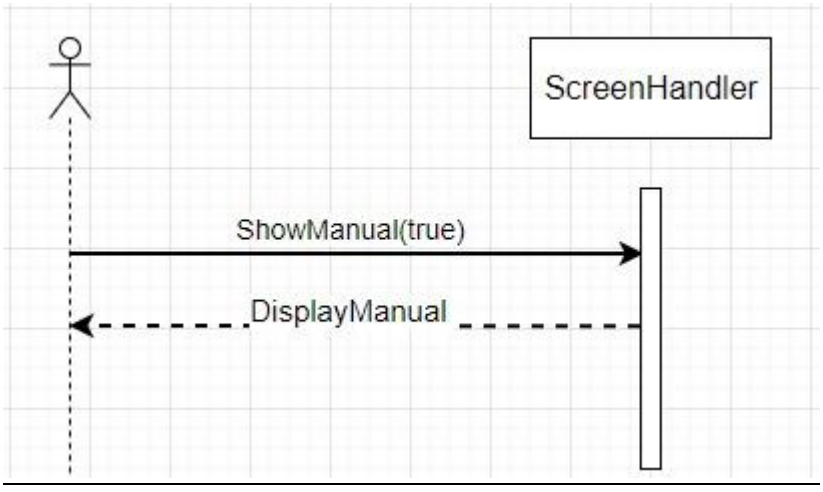


Figure 5.5

❖ 5.5- The Block Diagram for the proposed system:

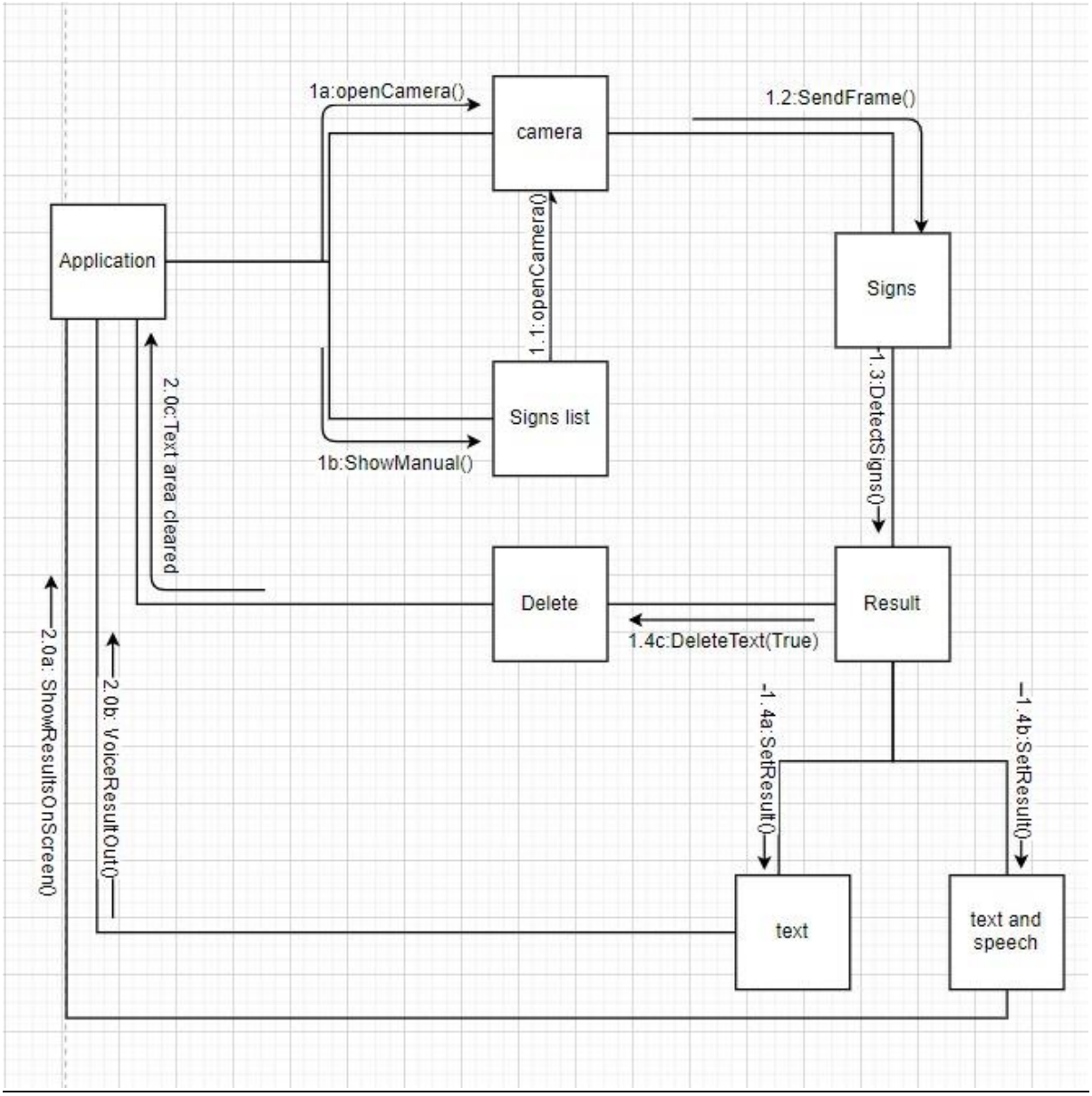
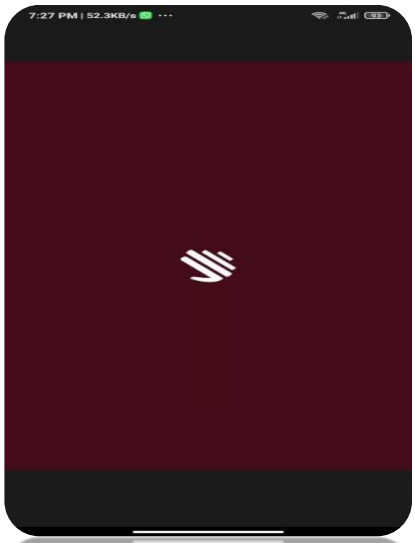


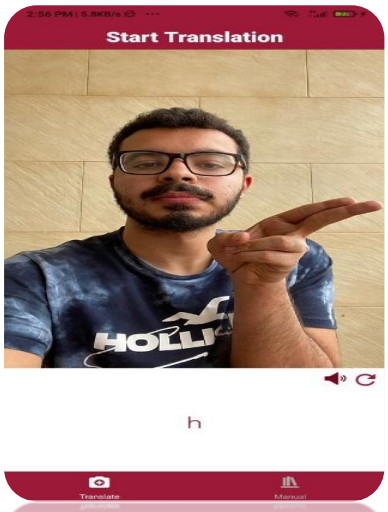
Figure 5.6

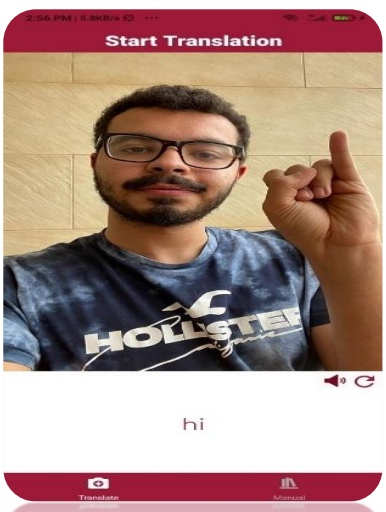
❖ 5.6- System GUI Design:

The following screens show a demo of how the application works. In the first screen we have the application icon after it was deployed showing the app's logo and name. The application then starts up showing the splash screen, then it opens to the home page with the camera open for translation. The user starts signing "hi there". In the 5th screen, it shows how the user can perform a 'space' sign to separate between two words. While the 11th screen shows how the user performs the 'delete' sign to delete the last character translated. The user can also clear the text area by pressing the refresh icon shown in the 12th screen. And the last screen shows the manual that the user can use as a guide while performing by pressing on the manual button down on the right. The sound option also exists if the user decides on enabling it for the text shown on the text area.

➢ 5.6.1- Prototype:

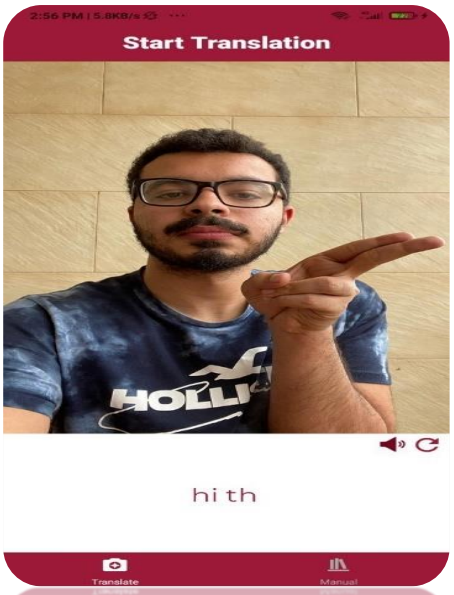




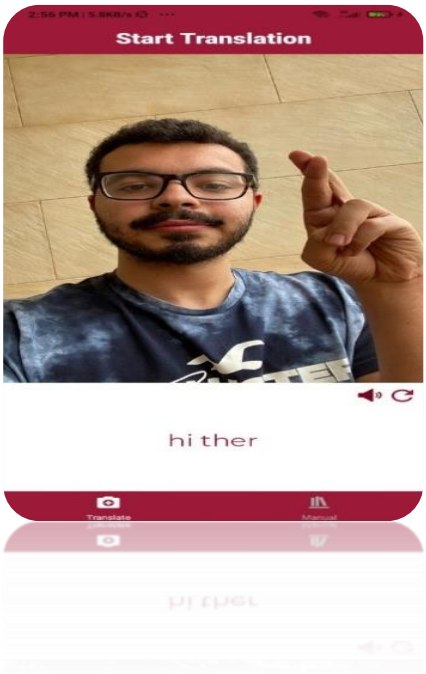


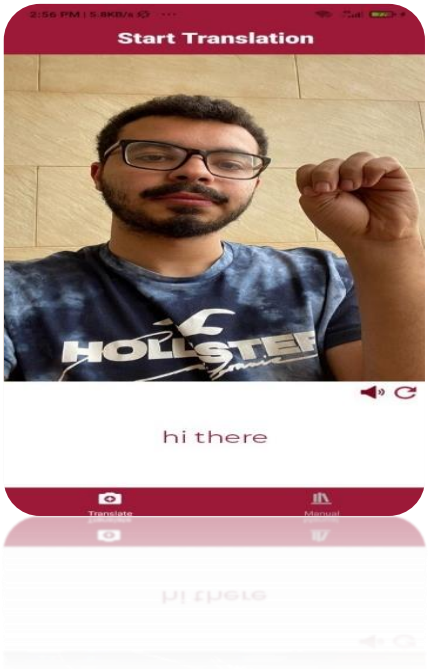

Start Translation
hi
Translate
Manual

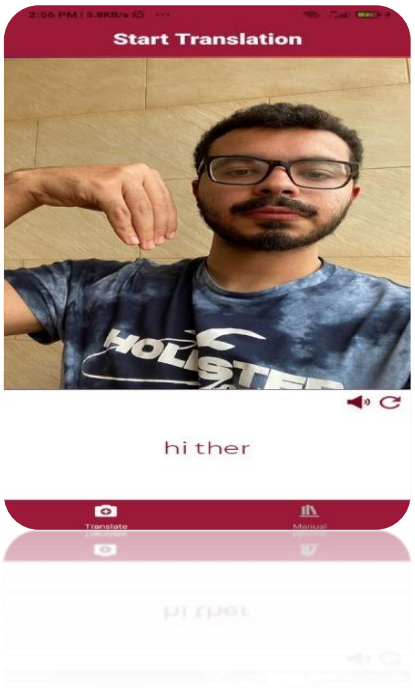

Start Translation
hi t
Translate
Manual

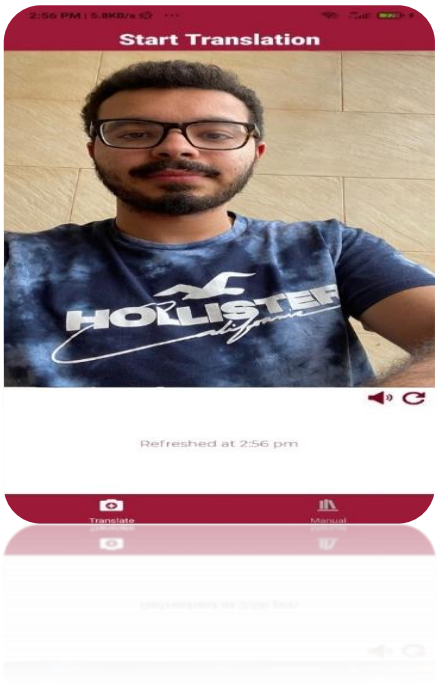

Start Translation
hi th
Translate
Manual


Start Translation
hi the
Translate
Manual


Start Translation
hi ther
Translate
Manual

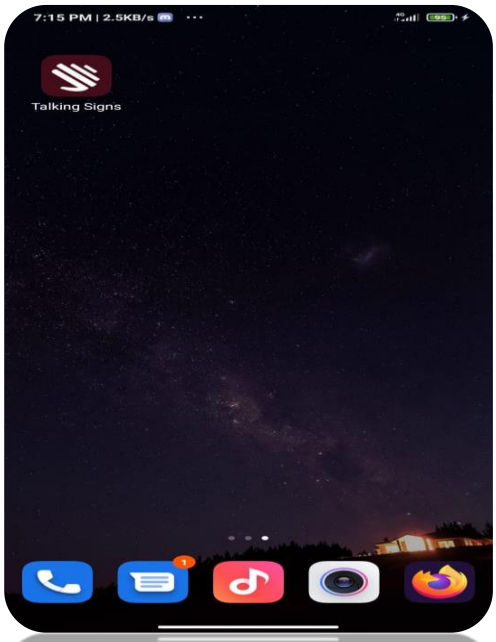

Start Translation
hi there
Translate
Manual

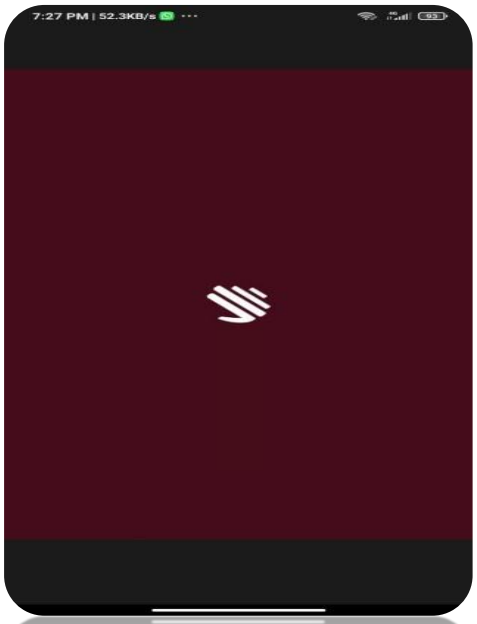

Start Translation
hi ther
Translate
Manual

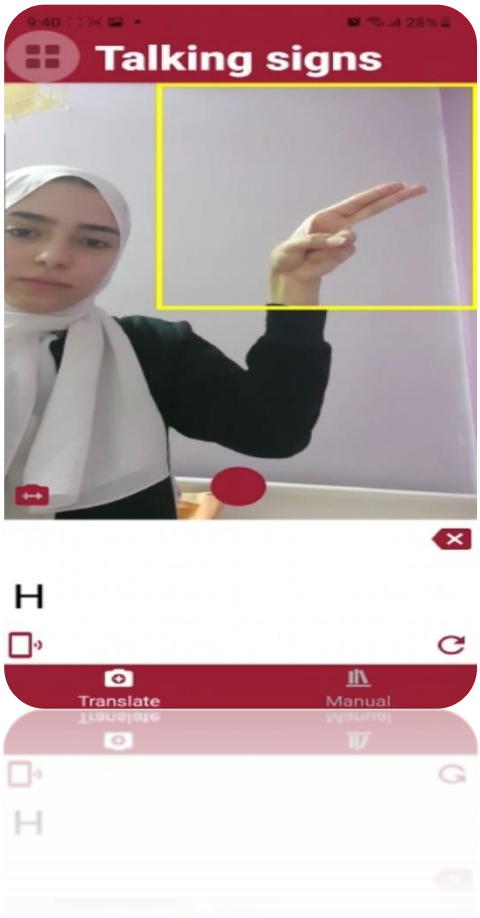

Start Translation
Refreshed at 2:56 pm
Translate
Manual

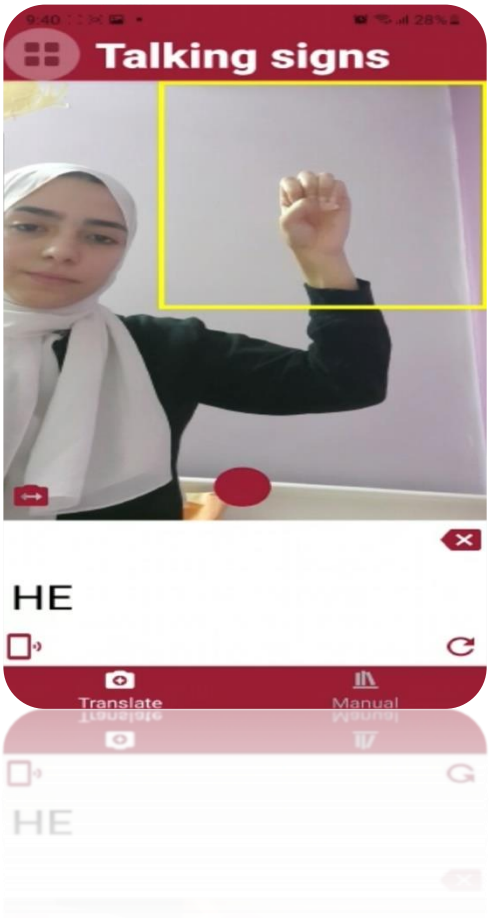

Manual
a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x
y
z
space
delete
Translate
Manual

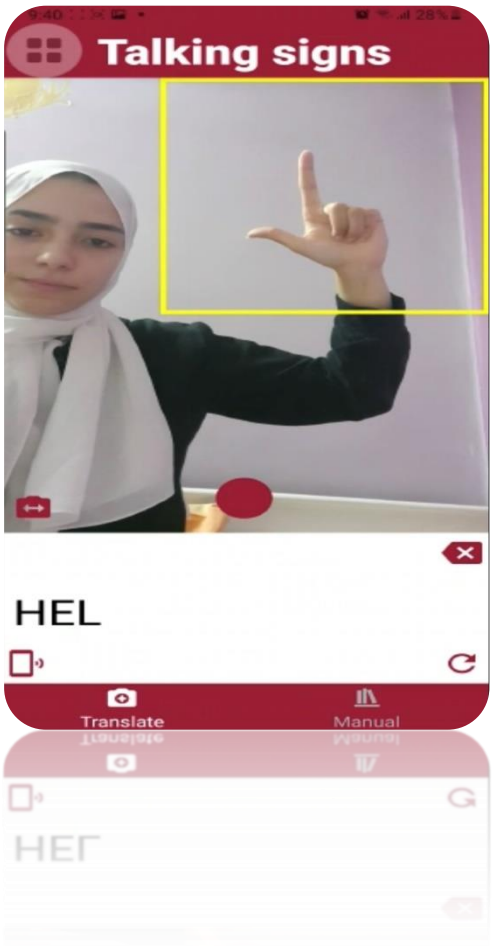
➢ 5.6.2- Final Real Version:

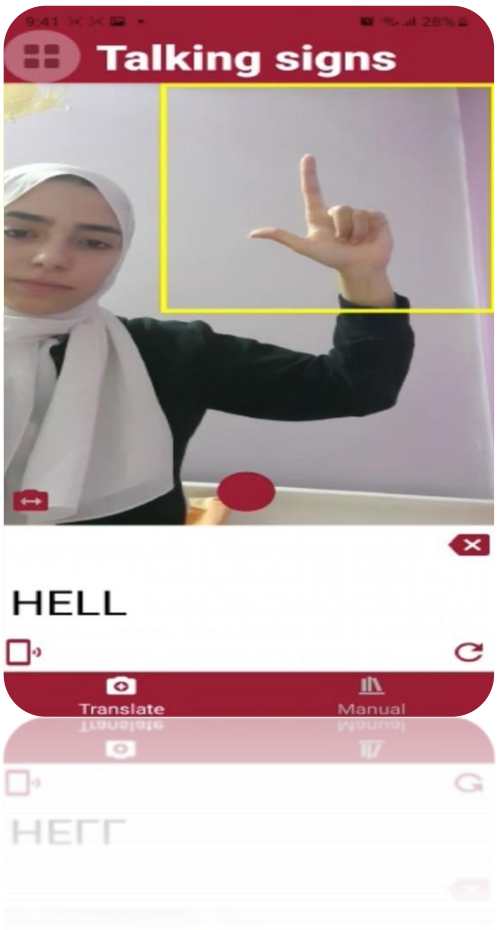


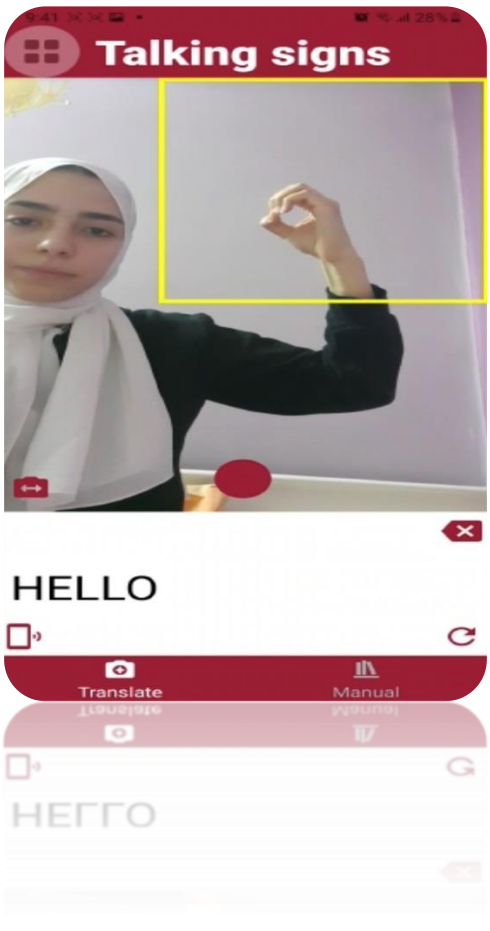


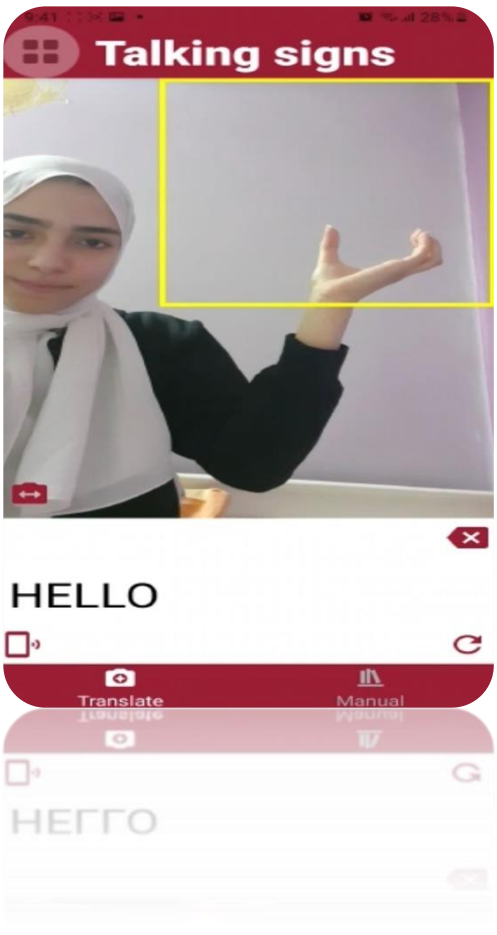


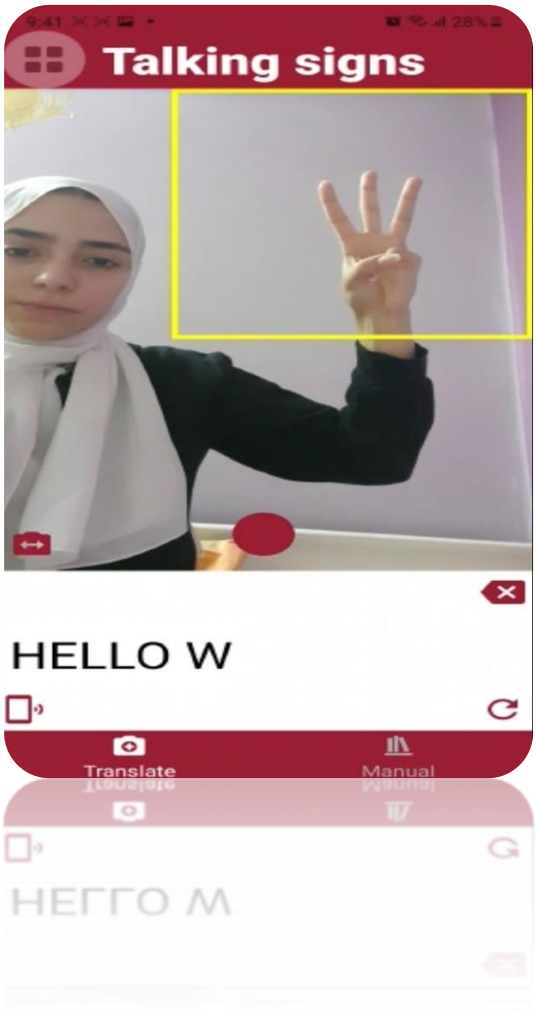


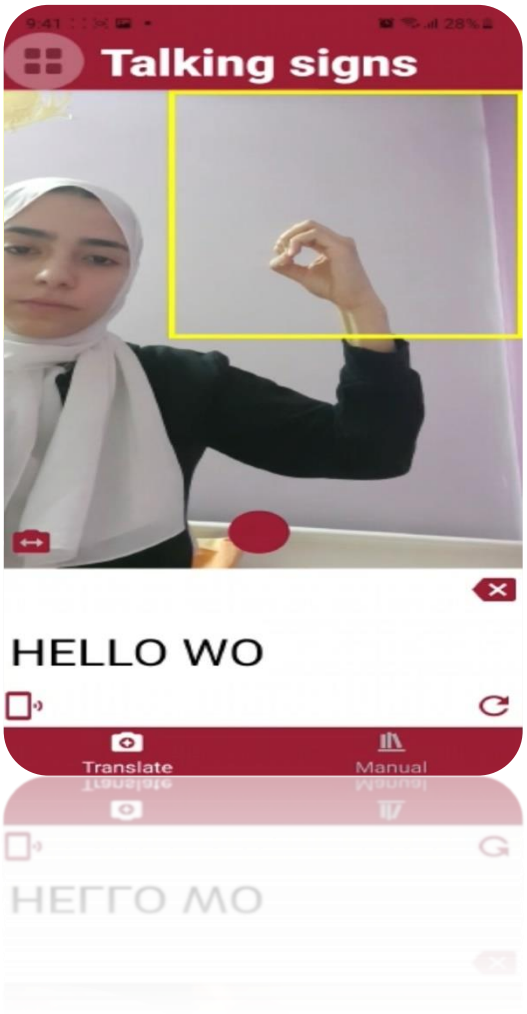

Talking signs
HEL
Translate
Manual

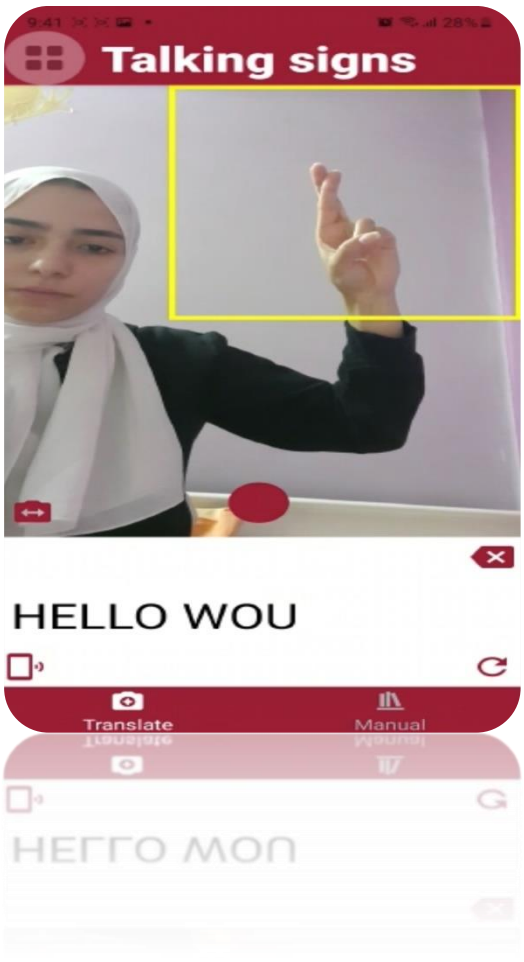

Talking signs
HELL
Translate
Manual

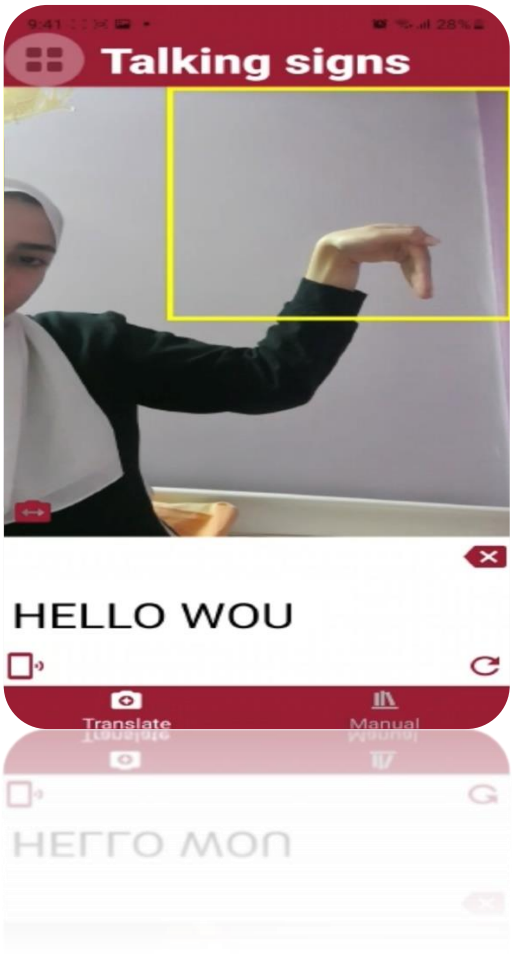

Talking signs
HELLO
Translate
Manual

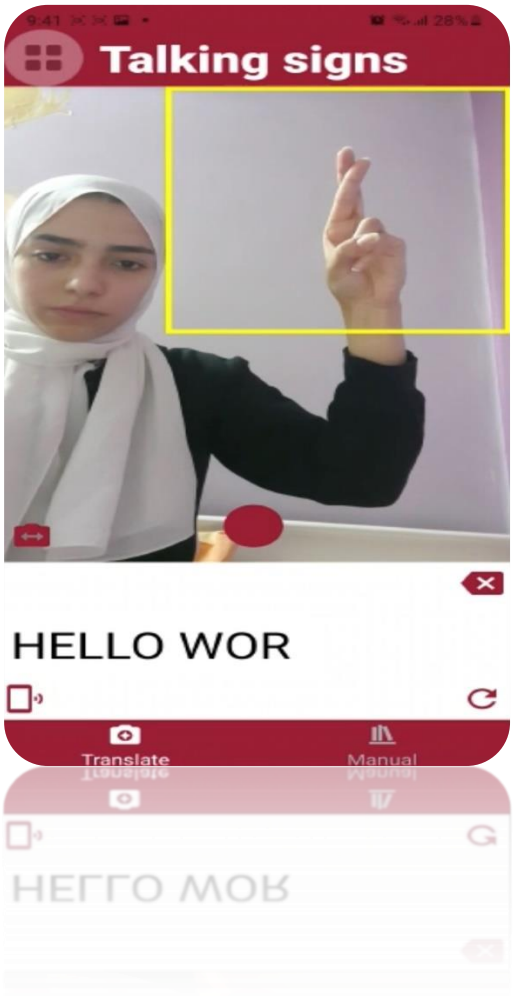

Talking signs
HELLO
Translate
Manual

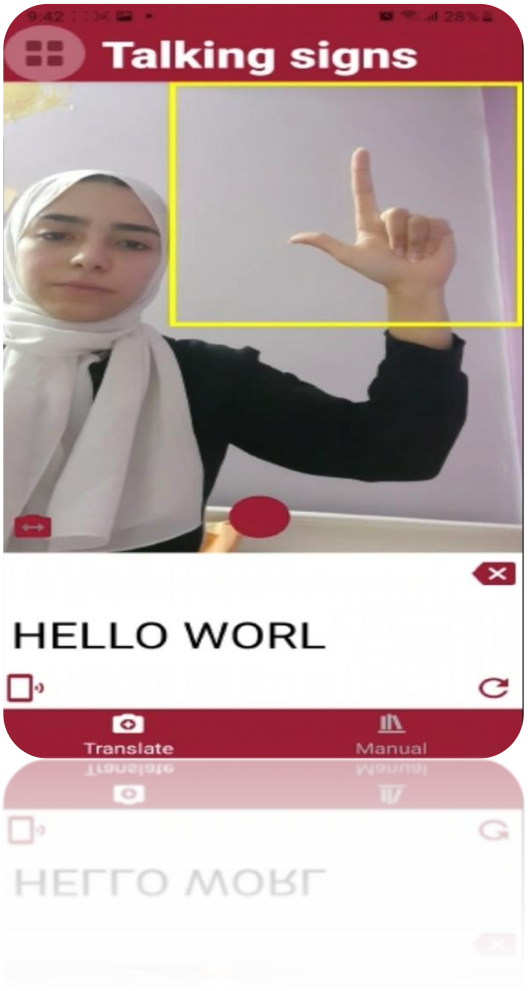

Talking signs
HELLO W
Translate
Manual

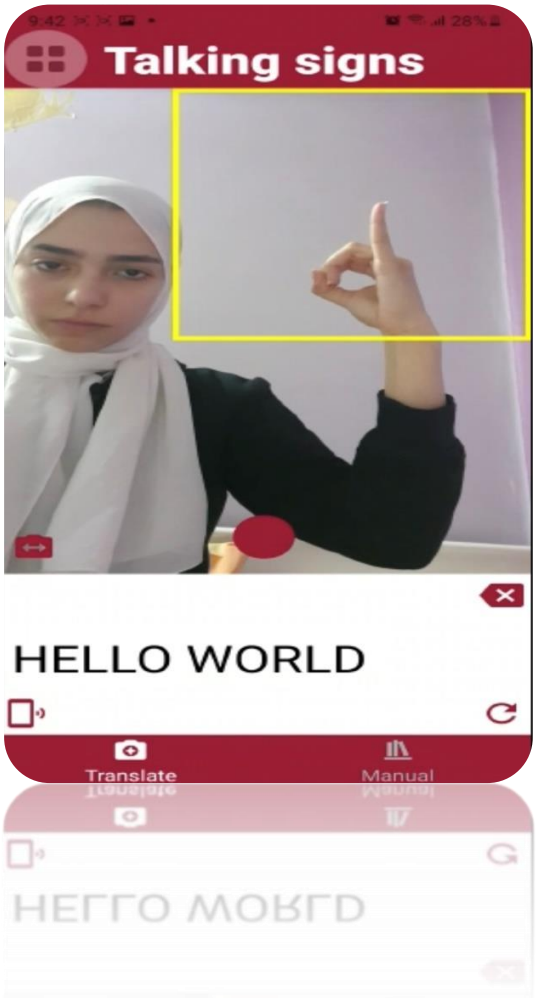

Talking signs
HELLO WO
Translate
Manual


Talking signs
HELLO WOU
Translate
Manual


Talking signs
HELLO WOU
Translate
Manual


Talking signs
HELLO WOR
Translate
Manual


Talking signs
HELLO WORL
Translate
Manual


Talking signs
HELLO WORLD
Translate
Manual


Manual
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
space
delete
Translate
Manual

Chapter 6: Implementation & Testing

❖ 6.1- Implementation:

➢ 6.1.1- CNN Model Development and Training:

➔ 6.1.1.1- Transfer Learning Approach:

- The Sign Language Detection model we are building is a real time Convolutional Neural Network (CNN) that's based on supervised learning algorithms. The approach followed to implement the Supervised Learning on our model is Transfer Learning, which is based on training a pre-trained model.
- A pre-trained model is a model trained on a large dataset and has learned general features and knowledge producing a weights file.
- To use the pre-trained model, we create our classifier model that consists of 1024 neurons, followed by another layer (which is the output layer) consisting of 29 neurons (number of classes we have), we then merge the pre-trained model with the model we developed by making the output of the pre-trained model as an input to our model; thus concatenating both as one model, where our developed model uses the weights generated from the pre-trained model as its initial weights.
- The advantage of using a pre-trained model is that it gives higher accuracy to the overall model; that's because the model would have already learned general features that will help in the classification process and the use of these features will not require high computational power.

➔ 6.1.1.2- Inception Model V3:

- There are different types of pre-trained models, but we used the Inception Version (3) model that was created by Google. This model is trained on the ImageNet dataset which consists of 14 million images, divided into 21 thousands classes. There are other pre-trained models that can be used such as MobileNet and VGG16, but the Inception V3 model has proved to provide more accuracy as it uses different types of filters that capture more knowledge from the pixels of an image compared to the others. Not to mention that the VGG16 when using filters in its convolutional layer, it filters each channel of the RGB (Red Channel, Green Channel and Blue Channel) separately and doesn't get the correlation between them; thereby missing vital knowledge that can cause a drop in the accuracy. This issue is handled in the Inception model as it filters each channel separately then gets the correlation among the three channels. As for the MobileNet, it has an advantage of having fewer parameters than the Inception model, which decreases its computational time. However, it affects its accuracy; as these parameters will have knowledge that can make the model learn better.
- The inception model consists of 311 layers, divided into 3 blocks. When using the inception model, we lock the first two blocks (250 layers out of the 311 layers), meaning that we don't retrain them; as they mostly will have general features that can be used as they are, and to also lower the computational power and time of the training. The last block (the remaining 61 layers) are the ones we retrain with the two concatenated layers we create (one containing 1024 neurons and the second contains 29 neurons); as the last block will contain the local features that will help in the classification process.

- The 311 layers that the Inception model has, are mainly 5 layers repeated in a sequence. First layer is the Input Layer that takes its input as images from the dataset we loaded. Next, we have the Conv2d Layer, this layer is responsible for using filters on the input image, and these filters are used to gain knowledge about the pixels. The values of these filters are what the model tries to deduce by training to achieve high accuracy as these are the weights that will be used when training is done. Filters can be small like 3x3 (producing 9 parameters) or large like 7x7 (producing 49 parameters), pros of using small filters is that it captures the little details that can make difference in the learning process of the model and takes less computational time, while the large features captures more knowledge concerning the relation between the pixels, but takes more computational time obviously. The Inception model V3 uses both types of filters in the perfect harmony to get both the small details and the relational knowledge between the pixels in a reasonable time while achieving high accuracy. The filters are applied on each channel of the RGB channels separately, then concatenated together to get the correlation between them in the next layer the Inception model has, which is the Mixed Layer. As for the Batch Normalization Layer, it is responsible for the data normalization that gets carried out on each batch that enters the training, knowing that we chose the batch size to be 64 images; this normalizes the data to eliminate extremes and let the data be within a certain range. Following we have the Activation Layer, this layer applies the activation function on the processed data, activation functions can be Relu or Softmax.

➔ **6.1.1.3- Dataset Used and Data Augmentation:**

- The Dataset we are training the model on, is provided by Kaggle and it consists of 87,000 images distributed over the 29 classes, which are the 26 alphabets and 3 classes for the nothing, space and delete signs. This Dataset also contains a testing sector which contains 28 images to test the model with after the training is done. We decided to take out 10% of the training dataset, which is 87,000 images, to validate the model with this 10% at the end of each epoch. This is done to prevent overfitting of the model, as this will let us keep track of the training accuracy in comparison with the validation accuracy. If the difference starts to increase between the two percentages, we would know that overfitting is happening and will stop the epoch running; saving us time from finding out after the whole training process is done. We applied the Data Augmentation technique on its images, this is done by causing variations in the images in terms of brightness shift and zoom range, so that it can train the model on more real time scenarios and images that were not noise filtered, this aims for better accuracy and helps in preventing overfitting. The data is augmented using brightness shift ranging in 20% darker lighting conditions and zooming out up to 120%. However the testing images provided are not sufficient to determine the testing accuracy of the model after training, so as a solution we created a dataset of our own, containing 125 images and coded a python script that passes these images to the model and calculates the accuracy.

➔ 6.1.1.4- The Training Process:

- The training process was carried out using Google Colab, took over three weeks to be completed; that's because of the session restriction by Google Colab which lets you train for only 8 hours, knowing that one epoch can take from one hour to one hour and a half while running for example 20 epochs, so it will definitely take more than 8 hours, making the session timing a limitation for us. In the beginning we started taking checkpoints every 20 or 30 epochs to prevent overfitting of the model and keep track on the accuracy achieved. Each epoch was designed to take 200 batches, each batch containing 64 images and the validation is done at the end of each epoch, taking 50 batches. The learning rate used is 0.0001 and the optimizer used to calculate the loss function and update the weights is the Stochastic Gradient Descent. Dropout technique also takes place after each epoch training; it aims to remove 20% of the filters existing in the model, so that the model does not start depending on specific filters that can therefore cause overfitting. Dropout technique is one of the overfitting prevention methods we used in our model.

➔ 6.1.15- Model Training on Google Colab:

Setting up the environment and kaggle API

Importing tensorflow and checking tensorflow

```
import tensorflow as tf

print(tf.__version__)
```

Installing kaggle so as to download the dataset using kaggle API:

```
!pip install -q kaggle
```

Setting up the kaggle.json authentication file enabling me to download the dataset:

```
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
```

Downloading the dataset using the API:

```
!kaggle datasets download -d grassknoted/asl-alphabet
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
!cd asl
```

Extracting the contents:

```
!unzip asl-alphabet.zip
```

Looking at the dataset

Specifying train and test directories:

```
# Specifying the training and test directories

TRAINING_DIR = './asl_alphabet_train/asl_alphabet_train/'
TEST_DIR = './asl_alphabet_test/asl_alphabet_test/'
```

Looking at some random images from the dataset:

```
# Printing 5 random images from any training category or from a specified category
%matplotlib inline

import cv2
import os
import random
import numpy as np
import matplotlib.image as mpimg
import matplotlib.pyplot as plt

number_of_rows = 1
number_of_columns = 5

categories = os.listdir(TRAINING_DIR)

random.seed(13)

category = categories[random.randint(1, 30)]
# category = 'A'

for i in range(number_of_columns):
  subplot = plt.subplot(number_of_rows, number_of_columns, i + 1)
  subplot.axis('Off')
  subplot.set_title(category)
  image_path = os.path.join(
      TRAINING_DIR,
      str(category),
      str(category) + str(random.randint(1, 1000)) + '.jpg'
  )
  image = mpimg.imread(image_path)
  plt.imshow(image)

plt.show()
```

Output of random images:

Preparing the training set

Augmenting the data with brightness and zoom ranges:

```
# Preparing ImageDataGenerator object for training the model
from tensorflow.keras.preprocessing.image import ImageDataGenerator

IMAGE_SIZE = 200
BATCH_SIZE = 64

data_generator = ImageDataGenerator(
    samplewise_center=True,
    samplewise_std_normalization=True,
    brightness_range=[0.8, 1.0],
    zoom_range=[1.0, 1.2],
    validation_split=0.1
)

train_generator = data_generator.flow_from_directory(TRAINING_DIR, target_size=(IMAGE_SIZE, IMAGE_SIZE), shuffle=True, seed=13,
                            class_mode='categorical', batch_size=BATCH_SIZE, subset="training")

validation_generator=data_generator.flow_from_directory(TRAINING_DIR, target_size=(IMAGE_SIZE, IMAGE_SIZE),shuffle=True,seed=13,
                                      class_mode='categorical', batch_size=BATCH_SIZE, subset="validation")
```

Preparing Inception V3 Network for transfer learning:

```
# Loading inception v3 network for transfer learning
from tensorflow.keras import layers
from tensorflow.keras import Model

from tensorflow.keras.applications.inception_v3 import InceptionV3

WEIGHTS_FILE = './inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5'

inception_v3_model = InceptionV3(
    input_shape = (IMAGE_SIZE, IMAGE_SIZE, 3),
    include_top = False,
    weights = 'imagenet'
)

# Not required --> inception_v3_model.load_weights(WEIGHTS_FILE)

# Enabling the top 2 inception blocks to train
for layer in inception_v3_model.layers[:249]:
    layer.trainable = False
for layer in inception_v3_model.layers[249:]:
    layer.trainable = True
```

Trying VGG16 Model then returning back to Inception V3:

```
# from keras.optimizers import SGD
# from keras.optimizers import Adam
# from tensorflow.keras.applications import VGG16
# from tensorflow.keras import models,layers,optimizers
# from tensorflow.keras.layers import Input, Dense, Reshape, Activation
# from tensorflow.keras.layers import  AveragePooling2D, ZeroPadding2D, Dropout, Flatten
# import keras as keras
# vgg_base = VGG16(weights='imagenet',include_top=False,input_shape=(IMAGE_SIZE,IMAGE_SIZE,3))

# #initiate a model
# model = models.Sequential()

# #Add the VGG base model
# model.add(vgg_base)

# #Add new layers
# model.add(layers.Flatten())

# model.add(Dense(8192, activation='relu'))
# model.add(Dropout(0.8))
# model.add(Dense(4096, activation='relu'))
# model.add(Dropout(0.5))
# model.add(Dense(29, activation='softmax'))

# #summary of the model
# #Adam=Adam(lr=0.0001, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, amsgrad=False)
# sgd = SGD(lr=0.001)
# model.compile(optimizer=sgd, loss='categorical_crossentropy', metrics=['accuracy'])
```

Choosing the inception output layer:

```
# Choosing the output layer to be merged with our FC layers (if required)
inception_output_layer = inception_v3_model.get_layer('mixed7')
print('Inception model output shape:', inception_output_layer.output_shape)

# Not required --> inception_output = inception_output_layer.output
inception_output = inception_v3_model.output
```

```
Inception model output shape: (None, 10, 10, 768)
```

Adding our own set of fully connected layers at the end of Inception v3 network:

```
from tensorflow.keras.optimizers import RMSprop, Adam, SGD

x = layers.GlobalAveragePooling2D()(inception_output)
x = layers.Dense(1024, activation='relu')(x)
# Not required -->
# model.add(Dense(8192, activation='relu'))
x = layers.Dropout(0.2)(x)
# model.add(Dense(4096, activation='relu'))
# model.add(Dropout(0.5))
x = layers.Dense(29, activation='softmax')(x)

model = Model(inception_v3_model.input, x)

model.compile(
    optimizer=SGD(learning_rate=0.0001, momentum=0.9),
    loss='categorical_crossentropy',
    metrics=['acc']
)
```

Looking at the final model:

```
# Watch the new model summary
model.summary()
```

Output of the final model:

```
Model: "model_1"
_________________________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
=================================================================================================
input_1 (InputLayer)         [(None, 200, 200, 3)] 0
_________________________________________________________________________________________________
conv2d (Conv2D)              (None, 99, 99, 32)   864         input_1[0][0]
_________________________________________________________________________________________________
batch_normalization (BatchNorma (None, 99, 99, 32)   96          conv2d[0][0]
_________________________________________________________________________________________________
activation (Activation)      (None, 99, 99, 32)   0           batch_normalization[0][0]
_________________________________________________________________________________________________
conv2d_1 (Conv2D)            (None, 97, 97, 32)   9216        activation[0][0]
_________________________________________________________________________________________________
batch_normalization_1 (BatchNor (None, 97, 97, 32)   96          conv2d_1[0][0]
_________________________________________________________________________________________________
activation_1 (Activation)    (None, 97, 97, 32)   0           batch_normalization_1[0][0]
_________________________________________________________________________________________________
conv2d_2 (Conv2D)            (None, 97, 97, 64)   18432       activation_1[0][0]
_________________________________________________________________________________________________
batch_normalization_2 (BatchNor (None, 97, 97, 64)   192         conv2d_2[0][0]
_________________________________________________________________________________________________
activation_2 (Activation)    (None, 97, 97, 64)   0           batch_normalization_2[0][0]
_________________________________________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 48, 48, 64)   0           activation_2[0][0]
_________________________________________________________________________________________________
conv2d_3 (Conv2D)            (None, 48, 48, 80)   5120        max_pooling2d[0][0]
_________________________________________________________________________________________________
batch_normalization_3 (BatchNor (None, 48, 48, 80)   240         conv2d_3[0][0]
_________________________________________________________________________________________________
activation_3 (Activation)    (None, 48, 48, 80)   0           batch_normalization_3[0][0]
_________________________________________________________________________________________________
conv2d_4 (Conv2D)            (None, 46, 46, 192)  138240      activation_3[0][0]
_________________________________________________________________________________________________
```

batch_normalization_4 (BatchNor	(None, 46, 46, 192)	576	conv2d_4[0][0]
activation_4 (Activation)	(None, 46, 46, 192)	0	batch_normalization_4[0][0]
max_pooling2d_1 (MaxPooling2D)	(None, 22, 22, 192)	0	activation_4[0][0]
conv2d_8 (Conv2D)	(None, 22, 22, 64)	12288	max_pooling2d_1[0][0]
batch_normalization_8 (BatchNor	(None, 22, 22, 64)	192	conv2d_8[0][0]
activation_8 (Activation)	(None, 22, 22, 64)	0	batch_normalization_8[0][0]
conv2d_6 (Conv2D)	(None, 22, 22, 48)	9216	max_pooling2d_1[0][0]
conv2d_9 (Conv2D)	(None, 22, 22, 96)	55296	activation_8[0][0]
batch_normalization_6 (BatchNor	(None, 22, 22, 48)	144	conv2d_6[0][0]
batch_normalization_9 (BatchNor	(None, 22, 22, 96)	288	conv2d_9[0][0]
activation_6 (Activation)	(None, 22, 22, 48)	0	batch_normalization_6[0][0]
activation_9 (Activation)	(None, 22, 22, 96)	0	batch_normalization_9[0][0]
average_pooling2d (AveragePooli	(None, 22, 22, 192)	0	max_pooling2d_1[0][0]
conv2d_5 (Conv2D)	(None, 22, 22, 64)	12288	max_pooling2d_1[0][0]
conv2d_7 (Conv2D)	(None, 22, 22, 64)	76800	activation_6[0][0]
conv2d_10 (Conv2D)	(None, 22, 22, 96)	82944	activation_9[0][0]
conv2d_11 (Conv2D)	(None, 22, 22, 32)	6144	average_pooling2d[0][0]
batch_normalization_5 (BatchNor	(None, 22, 22, 64)	192	conv2d_5[0][0]
batch_normalization_7 (BatchNor	(None, 22, 22, 64)	192	conv2d_7[0][0]
batch_normalization_10 (BatchNo	(None, 22, 22, 96)	288	conv2d_10[0][0]
batch_normalization_11 (BatchNo	(None, 22, 22, 32)	96	conv2d_11[0][0]
activation_5 (Activation)	(None, 22, 22, 64)	0	batch_normalization_5[0][0]
activation_7 (Activation)	(None, 22, 22, 64)	0	batch_normalization_7[0][0]
activation_10 (Activation)	(None, 22, 22, 96)	0	batch_normalization_10[0][0]
activation_11 (Activation)	(None, 22, 22, 32)	0	batch_normalization_11[0][0]
mixed0 (Concatenate)	(None, 22, 22, 256)	0	activation_5[0][0] activation_7[0][0] activation_10[0][0] activation_11[0][0]
conv2d_15 (Conv2D)	(None, 22, 22, 64)	16384	mixed0[0][0]
batch_normalization_15 (BatchNo	(None, 22, 22, 64)	192	conv2d_15[0][0]
activation_15 (Activation)	(None, 22, 22, 64)	0	batch_normalization_15[0][0]
conv2d_13 (Conv2D)	(None, 22, 22, 48)	12288	mixed0[0][0]
conv2d_16 (Conv2D)	(None, 22, 22, 96)	55296	activation_15[0][0]
batch_normalization_13 (BatchNo	(None, 22, 22, 48)	144	conv2d_13[0][0]
batch_normalization_16 (BatchNo	(None, 22, 22, 96)	288	conv2d_16[0][0]
activation_13 (Activation)	(None, 22, 22, 48)	0	batch_normalization_13[0][0]
activation_16 (Activation)	(None, 22, 22, 96)	0	batch_normalization_16[0][0]
average_pooling2d_1 (AveragePoo	(None, 22, 22, 256)	0	mixed0[0][0]
conv2d_12 (Conv2D)	(None, 22, 22, 64)	16384	mixed0[0][0]
conv2d_14 (Conv2D)	(None, 22, 22, 64)	76800	activation_13[0][0]
conv2d_17 (Conv2D)	(None, 22, 22, 96)	82944	activation_16[0][0]
conv2d_18 (Conv2D)	(None, 22, 22, 64)	16384	average_pooling2d_1[0][0]
batch_normalization_12 (BatchNo	(None, 22, 22, 64)	192	conv2d_12[0][0]
batch_normalization_14 (BatchNo	(None, 22, 22, 64)	192	conv2d_14[0][0]
batch_normalization_17 (BatchNo	(None, 22, 22, 96)	288	conv2d_17[0][0]
batch_normalization_18 (BatchNo	(None, 22, 22, 64)	192	conv2d_18[0][0]
activation_12 (Activation)	(None, 22, 22, 64)	0	batch_normalization_12[0][0]
activation_14 (Activation)	(None, 22, 22, 64)	0	batch_normalization_14[0][0]
activation_17 (Activation)	(None, 22, 22, 96)	0	batch_normalization_17[0][0]
activation_18 (Activation)	(None, 22, 22, 64)	0	batch_normalization_18[0][0]

mixed1 (Concatenate)	(None, 22, 22, 288)	0	activation_12[0][0] activation_14[0][0] activation_17[0][0] activation_18[0][0]
conv2d_22 (Conv2D)	(None, 22, 22, 64)	18432	mixed1[0][0]
batch_normalization_22 (BatchNo	(None, 22, 22, 64)	192	conv2d_22[0][0]
activation_22 (Activation)	(None, 22, 22, 64)	0	batch_normalization_22[0][0]
conv2d_20 (Conv2D)	(None, 22, 22, 48)	13824	mixed1[0][0]
conv2d_23 (Conv2D)	(None, 22, 22, 96)	55296	activation_22[0][0]
batch_normalization_20 (BatchNo	(None, 22, 22, 48)	144	conv2d_20[0][0]
batch_normalization_23 (BatchNo	(None, 22, 22, 96)	288	conv2d_23[0][0]
activation_20 (Activation)	(None, 22, 22, 48)	0	batch_normalization_20[0][0]
activation_23 (Activation)	(None, 22, 22, 96)	0	batch_normalization_23[0][0]
average_pooling2d_2 (AveragePoo	(None, 22, 22, 288)	0	mixed1[0][0]
conv2d_19 (Conv2D)	(None, 22, 22, 64)	18432	mixed1[0][0]
conv2d_21 (Conv2D)	(None, 22, 22, 64)	76800	activation_20[0][0]
conv2d_24 (Conv2D)	(None, 22, 22, 96)	82944	activation_23[0][0]
conv2d_25 (Conv2D)	(None, 22, 22, 64)	18432	average_pooling2d_2[0][0]
batch_normalization_19 (BatchNo	(None, 22, 22, 64)	192	conv2d_19[0][0]
batch_normalization_21 (BatchNo	(None, 22, 22, 64)	192	conv2d_21[0][0]
batch_normalization_24 (BatchNo	(None, 22, 22, 96)	288	conv2d_24[0][0]
batch_normalization_25 (BatchNo	(None, 22, 22, 64)	192	conv2d_25[0][0]
activation_19 (Activation)	(None, 22, 22, 64)	0	batch_normalization_19[0][0]
activation_21 (Activation)	(None, 22, 22, 64)	0	batch_normalization_21[0][0]
activation_24 (Activation)	(None, 22, 22, 96)	0	batch_normalization_24[0][0]
activation_25 (Activation)	(None, 22, 22, 64)	0	batch_normalization_25[0][0]
mixed2 (Concatenate)	(None, 22, 22, 288)	0	activation_19[0][0] activation_21[0][0] activation_24[0][0] activation_25[0][0]
conv2d_27 (Conv2D)	(None, 22, 22, 64)	18432	mixed2[0][0]
batch_normalization_27 (BatchNo	(None, 22, 22, 64)	192	conv2d_27[0][0]
activation_27 (Activation)	(None, 22, 22, 64)	0	batch_normalization_27[0][0]
conv2d_28 (Conv2D)	(None, 22, 22, 96)	55296	activation_27[0][0]
batch_normalization_28 (BatchNo	(None, 22, 22, 96)	288	conv2d_28[0][0]
activation_28 (Activation)	(None, 22, 22, 96)	0	batch_normalization_28[0][0]
conv2d_26 (Conv2D)	(None, 10, 10, 384)	995328	mixed2[0][0]
conv2d_29 (Conv2D)	(None, 10, 10, 96)	82944	activation_28[0][0]
batch_normalization_26 (BatchNo	(None, 10, 10, 384)	1152	conv2d_26[0][0]
batch_normalization_29 (BatchNo	(None, 10, 10, 96)	288	conv2d_29[0][0]
activation_26 (Activation)	(None, 10, 10, 384)	0	batch_normalization_26[0][0]
activation_29 (Activation)	(None, 10, 10, 96)	0	batch_normalization_29[0][0]
max_pooling2d_2 (MaxPooling2D)	(None, 10, 10, 288)	0	mixed2[0][0]
mixed3 (Concatenate)	(None, 10, 10, 768)	0	activation_26[0][0] activation_29[0][0] max_pooling2d_2[0][0]
conv2d_34 (Conv2D)	(None, 10, 10, 128)	98304	mixed3[0][0]
batch_normalization_34 (BatchNo	(None, 10, 10, 128)	384	conv2d_34[0][0]
activation_34 (Activation)	(None, 10, 10, 128)	0	batch_normalization_34[0][0]
conv2d_35 (Conv2D)	(None, 10, 10, 128)	114688	activation_34[0][0]
batch_normalization_35 (BatchNo	(None, 10, 10, 128)	384	conv2d_35[0][0]
activation_35 (Activation)	(None, 10, 10, 128)	0	batch_normalization_35[0][0]
conv2d_31 (Conv2D)	(None, 10, 10, 128)	98304	mixed3[0][0]
conv2d_36 (Conv2D)	(None, 10, 10, 128)	114688	activation_35[0][0]
batch_normalization_31 (BatchNo	(None, 10, 10, 128)	384	conv2d_31[0][0]

batch_normalization_36 (BatchNo	(None, 10, 10, 128)	384	conv2d_36[0][0]
activation_31 (Activation)	(None, 10, 10, 128)	0	batch_normalization_31[0][0]
activation_36 (Activation)	(None, 10, 10, 128)	0	batch_normalization_36[0][0]
conv2d_32 (Conv2D)	(None, 10, 10, 128)	114688	activation_31[0][0]
conv2d_37 (Conv2D)	(None, 10, 10, 128)	114688	activation_36[0][0]
batch_normalization_32 (BatchNo	(None, 10, 10, 128)	384	conv2d_32[0][0]
batch_normalization_37 (BatchNo	(None, 10, 10, 128)	384	conv2d_37[0][0]
activation_32 (Activation)	(None, 10, 10, 128)	0	batch_normalization_32[0][0]
activation_37 (Activation)	(None, 10, 10, 128)	0	batch_normalization_37[0][0]
average_pooling2d_3 (AveragePoo	(None, 10, 10, 768)	0	mixed3[0][0]
conv2d_30 (Conv2D)	(None, 10, 10, 192)	147456	mixed3[0][0]
conv2d_33 (Conv2D)	(None, 10, 10, 192)	172032	activation_32[0][0]
conv2d_38 (Conv2D)	(None, 10, 10, 192)	172032	activation_37[0][0]
conv2d_39 (Conv2D)	(None, 10, 10, 192)	147456	average_pooling2d_3[0][0]
batch_normalization_30 (BatchNo	(None, 10, 10, 192)	576	conv2d_30[0][0]
batch_normalization_33 (BatchNo	(None, 10, 10, 192)	576	conv2d_33[0][0]
batch_normalization_38 (BatchNo	(None, 10, 10, 192)	576	conv2d_38[0][0]
batch_normalization_39 (BatchNo	(None, 10, 10, 192)	576	conv2d_39[0][0]
activation_30 (Activation)	(None, 10, 10, 192)	0	batch_normalization_30[0][0]
activation_33 (Activation)	(None, 10, 10, 192)	0	batch_normalization_33[0][0]
activation_38 (Activation)	(None, 10, 10, 192)	0	batch_normalization_38[0][0]
activation_39 (Activation)	(None, 10, 10, 192)	0	batch_normalization_39[0][0]
mixed4 (Concatenate)	(None, 10, 10, 768)	0	activation_30[0][0] activation_33[0][0] activation_38[0][0] activation_39[0][0]

conv2d_44 (Conv2D)	(None, 10, 10, 160)	122880	mixed4[0][0]
batch_normalization_44 (BatchNo	(None, 10, 10, 160)	480	conv2d_44[0][0]
activation_44 (Activation)	(None, 10, 10, 160)	0	batch_normalization_44[0][0]
conv2d_45 (Conv2D)	(None, 10, 10, 160)	179200	activation_44[0][0]
batch_normalization_45 (BatchNo	(None, 10, 10, 160)	480	conv2d_45[0][0]
activation_45 (Activation)	(None, 10, 10, 160)	0	batch_normalization_45[0][0]
conv2d_41 (Conv2D)	(None, 10, 10, 160)	122880	mixed4[0][0]
conv2d_46 (Conv2D)	(None, 10, 10, 160)	179200	activation_45[0][0]
batch_normalization_41 (BatchNo	(None, 10, 10, 160)	480	conv2d_41[0][0]
batch_normalization_46 (BatchNo	(None, 10, 10, 160)	480	conv2d_46[0][0]
activation_41 (Activation)	(None, 10, 10, 160)	0	batch_normalization_41[0][0]
activation_46 (Activation)	(None, 10, 10, 160)	0	batch_normalization_46[0][0]
conv2d_42 (Conv2D)	(None, 10, 10, 160)	179200	activation_41[0][0]
conv2d_47 (Conv2D)	(None, 10, 10, 160)	179200	activation_46[0][0]
batch_normalization_42 (BatchNo	(None, 10, 10, 160)	480	conv2d_42[0][0]
batch_normalization_47 (BatchNo	(None, 10, 10, 160)	480	conv2d_47[0][0]
activation_42 (Activation)	(None, 10, 10, 160)	0	batch_normalization_42[0][0]
activation_47 (Activation)	(None, 10, 10, 160)	0	batch_normalization_47[0][0]
average_pooling2d_4 (AveragePoo	(None, 10, 10, 768)	0	mixed4[0][0]
conv2d_40 (Conv2D)	(None, 10, 10, 192)	147456	mixed4[0][0]
conv2d_43 (Conv2D)	(None, 10, 10, 192)	215040	activation_42[0][0]
conv2d_48 (Conv2D)	(None, 10, 10, 192)	215040	activation_47[0][0]
conv2d_49 (Conv2D)	(None, 10, 10, 192)	147456	average_pooling2d_4[0][0]
batch_normalization_40 (BatchNo	(None, 10, 10, 192)	576	conv2d_40[0][0]
batch_normalization_43 (BatchNo	(None, 10, 10, 192)	576	conv2d_43[0][0]

batch_normalization_48 (BatchNo	(None, 10, 10, 192)	576	conv2d_48[0][0]
batch_normalization_49 (BatchNo	(None, 10, 10, 192)	576	conv2d_49[0][0]
activation_40 (Activation)	(None, 10, 10, 192)	0	batch_normalization_40[0][0]
activation_43 (Activation)	(None, 10, 10, 192)	0	batch_normalization_43[0][0]
activation_48 (Activation)	(None, 10, 10, 192)	0	batch_normalization_48[0][0]
activation_49 (Activation)	(None, 10, 10, 192)	0	batch_normalization_49[0][0]
mixed5 (Concatenate)	(None, 10, 10, 768)	0	activation_40[0][0] activation_43[0][0] activation_48[0][0] activation_49[0][0]
conv2d_54 (Conv2D)	(None, 10, 10, 160)	122880	mixed5[0][0]
batch_normalization_54 (BatchNo	(None, 10, 10, 160)	480	conv2d_54[0][0]
activation_54 (Activation)	(None, 10, 10, 160)	0	batch_normalization_54[0][0]
conv2d_55 (Conv2D)	(None, 10, 10, 160)	179200	activation_54[0][0]
batch_normalization_55 (BatchNo	(None, 10, 10, 160)	480	conv2d_55[0][0]
activation_55 (Activation)	(None, 10, 10, 160)	0	batch_normalization_55[0][0]
conv2d_51 (Conv2D)	(None, 10, 10, 160)	122880	mixed5[0][0]
conv2d_56 (Conv2D)	(None, 10, 10, 160)	179200	activation_55[0][0]
batch_normalization_51 (BatchNo	(None, 10, 10, 160)	480	conv2d_51[0][0]
batch_normalization_56 (BatchNo	(None, 10, 10, 160)	480	conv2d_56[0][0]
activation_51 (Activation)	(None, 10, 10, 160)	0	batch_normalization_51[0][0]
activation_56 (Activation)	(None, 10, 10, 160)	0	batch_normalization_56[0][0]
conv2d_52 (Conv2D)	(None, 10, 10, 160)	179200	activation_51[0][0]
conv2d_57 (Conv2D)	(None, 10, 10, 160)	179200	activation_56[0][0]
batch_normalization_52 (BatchNo	(None, 10, 10, 160)	480	conv2d_52[0][0]
batch_normalization_57 (BatchNo	(None, 10, 10, 160)	480	conv2d_57[0][0]
activation_52 (Activation)	(None, 10, 10, 160)	0	batch_normalization_52[0][0]
activation_57 (Activation)	(None, 10, 10, 160)	0	batch_normalization_57[0][0]

average_pooling2d_5 (AveragePoo	(None, 10, 10, 768)	0	mixed5[0][0]
conv2d_50 (Conv2D)	(None, 10, 10, 192)	147456	mixed5[0][0]
conv2d_53 (Conv2D)	(None, 10, 10, 192)	215040	activation_52[0][0]
conv2d_58 (Conv2D)	(None, 10, 10, 192)	215040	activation_57[0][0]
conv2d_59 (Conv2D)	(None, 10, 10, 192)	147456	average_pooling2d_5[0][0]
batch_normalization_50 (BatchNo	(None, 10, 10, 192)	576	conv2d_50[0][0]
batch_normalization_53 (BatchNo	(None, 10, 10, 192)	576	conv2d_53[0][0]
batch_normalization_58 (BatchNo	(None, 10, 10, 192)	576	conv2d_58[0][0]
batch_normalization_59 (BatchNo	(None, 10, 10, 192)	576	conv2d_59[0][0]
activation_50 (Activation)	(None, 10, 10, 192)	0	batch_normalization_50[0][0]
activation_53 (Activation)	(None, 10, 10, 192)	0	batch_normalization_53[0][0]
activation_58 (Activation)	(None, 10, 10, 192)	0	batch_normalization_58[0][0]
activation_59 (Activation)	(None, 10, 10, 192)	0	batch_normalization_59[0][0]
mixed6 (Concatenate)	(None, 10, 10, 768)	0	activation_50[0][0] activation_53[0][0] activation_58[0][0] activation_59[0][0]
conv2d_64 (Conv2D)	(None, 10, 10, 192)	147456	mixed6[0][0]
batch_normalization_64 (BatchNo	(None, 10, 10, 192)	576	conv2d_64[0][0]
activation_64 (Activation)	(None, 10, 10, 192)	0	batch_normalization_64[0][0]
conv2d_65 (Conv2D)	(None, 10, 10, 192)	258048	activation_64[0][0]
batch_normalization_65 (BatchNo	(None, 10, 10, 192)	576	conv2d_65[0][0]
activation_65 (Activation)	(None, 10, 10, 192)	0	batch_normalization_65[0][0]
conv2d_61 (Conv2D)	(None, 10, 10, 192)	147456	mixed6[0][0]
conv2d_66 (Conv2D)	(None, 10, 10, 192)	258048	activation_65[0][0]
batch_normalization_61 (BatchNo	(None, 10, 10, 192)	576	conv2d_61[0][0]
batch_normalization_66 (BatchNo	(None, 10, 10, 192)	576	conv2d_66[0][0]

conv2d_62 (Conv2D)	(None, 10, 10, 192)	258048	activation_61[0][0]
conv2d_67 (Conv2D)	(None, 10, 10, 192)	258048	activation_66[0][0]
batch_normalization_62 (BatchNo	(None, 10, 10, 192)	576	conv2d_62[0][0]
batch_normalization_67 (BatchNo	(None, 10, 10, 192)	576	conv2d_67[0][0]
activation_62 (Activation)	(None, 10, 10, 192)	0	batch_normalization_62[0][0]
activation_67 (Activation)	(None, 10, 10, 192)	0	batch_normalization_67[0][0]
average_pooling2d_6 (AveragePoo	(None, 10, 10, 768)	0	mixed6[0][0]
conv2d_60 (Conv2D)	(None, 10, 10, 192)	147456	mixed6[0][0]
conv2d_63 (Conv2D)	(None, 10, 10, 192)	258048	activation_62[0][0]
conv2d_68 (Conv2D)	(None, 10, 10, 192)	258048	activation_67[0][0]
conv2d_69 (Conv2D)	(None, 10, 10, 192)	147456	average_pooling2d_6[0][0]
batch_normalization_60 (BatchNo	(None, 10, 10, 192)	576	conv2d_60[0][0]
batch_normalization_63 (BatchNo	(None, 10, 10, 192)	576	conv2d_63[0][0]
batch_normalization_68 (BatchNo	(None, 10, 10, 192)	576	conv2d_68[0][0]
batch_normalization_69 (BatchNo	(None, 10, 10, 192)	576	conv2d_69[0][0]
activation_60 (Activation)	(None, 10, 10, 192)	0	batch_normalization_60[0][0]
activation_63 (Activation)	(None, 10, 10, 192)	0	batch_normalization_63[0][0]
activation_68 (Activation)	(None, 10, 10, 192)	0	batch_normalization_68[0][0]
activation_69 (Activation)	(None, 10, 10, 192)	0	batch_normalization_69[0][0]
mixed7 (Concatenate)	(None, 10, 10, 768)	0	activation_60[0][0] activation_63[0][0] activation_68[0][0] activation_69[0][0]
conv2d_72 (Conv2D)	(None, 10, 10, 192)	147456	mixed7[0][0]
batch_normalization_72 (BatchNo	(None, 10, 10, 192)	576	conv2d_72[0][0]
activation_72 (Activation)	(None, 10, 10, 192)	0	batch_normalization_72[0][0]
conv2d_73 (Conv2D)	(None, 10, 10, 192)	258048	activation_72[0][0]
batch_normalization_73 (BatchNo	(None, 10, 10, 192)	576	conv2d_73[0][0]
activation_73 (Activation)	(None, 10, 10, 192)	0	batch_normalization_73[0][0]
conv2d_70 (Conv2D)	(None, 10, 10, 192)	147456	mixed7[0][0]
conv2d_74 (Conv2D)	(None, 10, 10, 192)	258048	activation_73[0][0]
batch_normalization_70 (BatchNo	(None, 10, 10, 192)	576	conv2d_70[0][0]
batch_normalization_74 (BatchNo	(None, 10, 10, 192)	576	conv2d_74[0][0]
activation_70 (Activation)	(None, 10, 10, 192)	0	batch_normalization_70[0][0]
activation_74 (Activation)	(None, 10, 10, 192)	0	batch_normalization_74[0][0]
conv2d_71 (Conv2D)	(None, 4, 4, 320)	552960	activation_70[0][0]
conv2d_75 (Conv2D)	(None, 4, 4, 192)	331776	activation_74[0][0]
batch_normalization_71 (BatchNo	(None, 4, 4, 320)	960	conv2d_71[0][0]
batch_normalization_75 (BatchNo	(None, 4, 4, 192)	576	conv2d_75[0][0]
activation_71 (Activation)	(None, 4, 4, 320)	0	batch_normalization_71[0][0]
activation_75 (Activation)	(None, 4, 4, 192)	0	batch_normalization_75[0][0]
max_pooling2d_3 (MaxPooling2D)	(None, 4, 4, 768)	0	mixed7[0][0]
mixed8 (Concatenate)	(None, 4, 4, 1280)	0	activation_71[0][0] activation_75[0][0] max_pooling2d_3[0][0]
conv2d_80 (Conv2D)	(None, 4, 4, 448)	573440	mixed8[0][0]
batch_normalization_80 (BatchNo	(None, 4, 4, 448)	1344	conv2d_80[0][0]
activation_80 (Activation)	(None, 4, 4, 448)	0	batch_normalization_80[0][0]
conv2d_77 (Conv2D)	(None, 4, 4, 384)	491520	mixed8[0][0]
conv2d_81 (Conv2D)	(None, 4, 4, 384)	1548288	activation_80[0][0]
batch_normalization_77 (BatchNo	(None, 4, 4, 384)	1152	conv2d_77[0][0]
batch_normalization_81 (BatchNo	(None, 4, 4, 384)	1152	conv2d_81[0][0]

conv2d_78 (Conv2D)	(None, 4, 4, 384)	442368	activation_77[0][0]
conv2d_79 (Conv2D)	(None, 4, 4, 384)	442368	activation_77[0][0]
conv2d_82 (Conv2D)	(None, 4, 4, 384)	442368	activation_81[0][0]
conv2d_83 (Conv2D)	(None, 4, 4, 384)	442368	activation_81[0][0]
average_pooling2d_7 (AveragePoo	(None, 4, 4, 1280)	0	mixed8[0][0]
conv2d_76 (Conv2D)	(None, 4, 4, 320)	409600	mixed8[0][0]
batch_normalization_78 (BatchNo	(None, 4, 4, 384)	1152	conv2d_78[0][0]
batch_normalization_79 (BatchNo	(None, 4, 4, 384)	1152	conv2d_79[0][0]
batch_normalization_82 (BatchNo	(None, 4, 4, 384)	1152	conv2d_82[0][0]
batch_normalization_83 (BatchNo	(None, 4, 4, 384)	1152	conv2d_83[0][0]
conv2d_84 (Conv2D)	(None, 4, 4, 192)	245760	average_pooling2d_7[0][0]
batch_normalization_76 (BatchNo	(None, 4, 4, 320)	960	conv2d_76[0][0]
activation_78 (Activation)	(None, 4, 4, 384)	0	batch_normalization_78[0][0]
activation_79 (Activation)	(None, 4, 4, 384)	0	batch_normalization_79[0][0]
activation_82 (Activation)	(None, 4, 4, 384)	0	batch_normalization_82[0][0]
activation_83 (Activation)	(None, 4, 4, 384)	0	batch_normalization_83[0][0]
batch_normalization_84 (BatchNo	(None, 4, 4, 192)	576	conv2d_84[0][0]
activation_76 (Activation)	(None, 4, 4, 320)	0	batch_normalization_76[0][0]
mixed9_0 (Concatenate)	(None, 4, 4, 768)	0	activation_78[0][0] activation_79[0][0]
concatenate (Concatenate)	(None, 4, 4, 768)	0	activation_82[0][0] activation_83[0][0]
activation_84 (Activation)	(None, 4, 4, 192)	0	batch_normalization_84[0][0]
mixed9 (Concatenate)	(None, 4, 4, 2048)	0	activation_76[0][0] mixed9_0[0][0] concatenate[0][0] activation_84[0][0]
conv2d_89 (Conv2D)	(None, 4, 4, 448)	917504	mixed9[0][0]
batch_normalization_89 (BatchNo	(None, 4, 4, 448)	1344	conv2d_89[0][0]
activation_89 (Activation)	(None, 4, 4, 448)	0	batch_normalization_89[0][0]
conv2d_86 (Conv2D)	(None, 4, 4, 384)	786432	mixed9[0][0]
conv2d_90 (Conv2D)	(None, 4, 4, 384)	1548288	activation_89[0][0]
batch_normalization_86 (BatchNo	(None, 4, 4, 384)	1152	conv2d_86[0][0]
batch_normalization_90 (BatchNo	(None, 4, 4, 384)	1152	conv2d_90[0][0]
activation_86 (Activation)	(None, 4, 4, 384)	0	batch_normalization_86[0][0]
activation_90 (Activation)	(None, 4, 4, 384)	0	batch_normalization_90[0][0]
conv2d_87 (Conv2D)	(None, 4, 4, 384)	442368	activation_86[0][0]
conv2d_88 (Conv2D)	(None, 4, 4, 384)	442368	activation_86[0][0]
conv2d_91 (Conv2D)	(None, 4, 4, 384)	442368	activation_90[0][0]
conv2d_92 (Conv2D)	(None, 4, 4, 384)	442368	activation_90[0][0]
average_pooling2d_8 (AveragePoo	(None, 4, 4, 2048)	0	mixed9[0][0]
conv2d_85 (Conv2D)	(None, 4, 4, 320)	655360	mixed9[0][0]
batch_normalization_87 (BatchNo	(None, 4, 4, 384)	1152	conv2d_87[0][0]
batch_normalization_88 (BatchNo	(None, 4, 4, 384)	1152	conv2d_88[0][0]
batch_normalization_91 (BatchNo	(None, 4, 4, 384)	1152	conv2d_91[0][0]
batch_normalization_92 (BatchNo	(None, 4, 4, 384)	1152	conv2d_92[0][0]
conv2d_93 (Conv2D)	(None, 4, 4, 192)	393216	average_pooling2d_8[0][0]
batch_normalization_85 (BatchNo	(None, 4, 4, 320)	960	conv2d_85[0][0]
activation_87 (Activation)	(None, 4, 4, 384)	0	batch_normalization_87[0][0]
activation_88 (Activation)	(None, 4, 4, 384)	0	batch_normalization_88[0][0]
activation_91 (Activation)	(None, 4, 4, 384)	0	batch_normalization_91[0][0]
activation_92 (Activation)	(None, 4, 4, 384)	0	batch_normalization_92[0][0]
batch_normalization_93 (BatchNo	(None, 4, 4, 192)	576	conv2d_93[0][0]

```
activation_85 (Activation)      (None, 4, 4, 320)    0           batch_normalization_85[0][0]
__________________________________________________________________________________________________
mixed9_1 (Concatenate)          (None, 4, 4, 768)    0           activation_87[0][0]
                                                                 activation_88[0][0]
__________________________________________________________________________________________________
concatenate_1 (Concatenate)     (None, 4, 4, 768)    0           activation_91[0][0]
                                                                 activation_92[0][0]
__________________________________________________________________________________________________
activation_93 (Activation)      (None, 4, 4, 192)    0           batch_normalization_93[0][0]
__________________________________________________________________________________________________
mixed10 (Concatenate)           (None, 4, 4, 2048)   0           activation_85[0][0]
                                                                 mixed9_1[0][0]
                                                                 concatenate_1[0][0]
                                                                 activation_93[0][0]
__________________________________________________________________________________________________
global_average_pooling2d_2 (Glo (None, 2048)         0           mixed10[0][0]
__________________________________________________________________________________________________
dense_3 (Dense)                 (None, 1024)         2098176     global_average_pooling2d_2[0][0]
__________________________________________________________________________________________________
dropout (Dropout)               (None, 1024)         0           dense_3[0][0]
__________________________________________________________________________________________________
dense_4 (Dense)                 (None, 29)           29725       dropout[0][0]
==================================================================================================
Total params: 23,930,685
Trainable params: 13,242,781
Non-trainable params: 10,687,904
```

Setting up a callback funtion in order to stop training at a particular threshold:

```
# Creating a callback to stop model training after reaching a threshold accuracy

LOSS_THRESHOLD = 0.2
ACCURACY_THRESHOLD = 0.95

class ModelCallback(tf.keras.callbacks.Callback):
  def on_epoch_end(self, epoch, logs={}):
    if logs.get('val_loss') <= LOSS_THRESHOLD and logs.get('val_acc') >= ACCURACY_THRESHOLD:
      print("\nReached", ACCURACY_THRESHOLD * 100, "accuracy, Stopping!")
      self.model.stop_training = True

callback = ModelCallback()
```

Training the model generated using Inception v3 and our own set of Fully Connected layers:

Fitting the model to the training dataset:

```
history = model.fit(
    train_generator,
    validation_data=validation_generator,
    steps_per_epoch=200,
    validation_steps=50,
    epochs=80,
    callbacks=[callback]
)
```

Plotting the results

Training Accuracy vs Validation Accuracy:

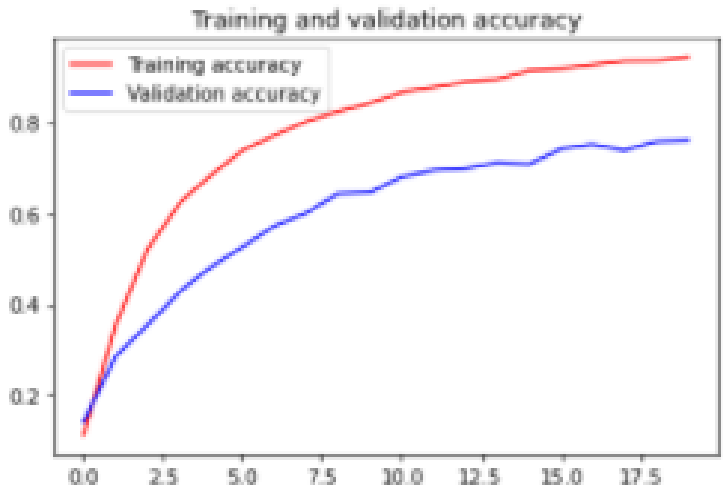
```
import matplotlib.pyplot as plt
acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(acc))

plt.plot(epochs, acc, 'r', label='Training accuracy')
plt.plot(epochs, val_acc, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.legend(loc=0)
plt.figure()


plt.show()
```

Plotting Result:



Training Loss vs Validation Loss

```
plt.plot(epochs, loss, 'r', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend(loc=0)
plt.figure()
```

Plotting Result:

Saving the model

As we were satisfied with our results we save our model:

```
# Saving the model
MODEL_NAME = 'models/asl_alphabet_{}.h5'.format(9575)
model.save(MODEL_NAME)
```

Testing our model

Calculating test accuracy:

```
import os as os
import cv2 as cv2
import numpy as np
test_images = os.listdir(TEST_DIR)
total_test_cases = len(test_images)
total_correctly_classified = 0
total_misclassified = 0
for i, test_image in enumerate(test_images):
    image_location = TEST_DIR + test_image
    img = cv2.imread(image_location)
    img = cv2.resize(img, (IMAGE_SIZE, IMAGE_SIZE))
    img = np.array(img) / 255.
    img = img.reshape((1, IMAGE_SIZE, IMAGE_SIZE, 3))
    img = data_generator.standardize(img)
    prediction = np.array(model.predict(img))
    actual = test_image.split('_')[0]
    predicted = classes[prediction.argmax()]
    print('Actual class: {} - Predicted class: {}'.format(
        actual, predicted), end=' ')
    if actual == predicted:
      print('PASS!')
      total_correctly_classified += 1
    else:
      print('FAIL!')
      total_misclassified += 1
print("=" * 20)
test_accuracy = (total_correctly_classified / total_test_cases) * 100
test_error_rate = (total_misclassified / total_test_cases) * 100

print('Test accuracy (%):', test_accuracy)
print('Test error rate (%):', test_error_rate)
print('Number of misclassified classes:', total_misclassified)
print('Number of correctly classified classes', total_correctly_classified)
```

Test accuracy Results:

```
Actual class: V - Predicted class: V PASS!
Actual class: F - Predicted class: F PASS!
Actual class: R - Predicted class: R PASS!
Actual class: H - Predicted class: H PASS!
Actual class: N - Predicted class: N PASS!
Actual class: O - Predicted class: O PASS!
Actual class: E - Predicted class: E PASS!
Actual class: B - Predicted class: B PASS!
Actual class: M - Predicted class: M PASS!
Actual class: A - Predicted class: A PASS!
Actual class: nothing - Predicted class: nothing PASS!
Actual class: D - Predicted class: D PASS!
Actual class: W - Predicted class: W PASS!
Actual class: Z - Predicted class: Z PASS!
Actual class: K - Predicted class: V FAIL!
Actual class: Y - Predicted class: Y PASS!
Actual class: L - Predicted class: L PASS!
Actual class: P - Predicted class: P PASS!
Actual class: U - Predicted class: B FAIL!
Actual class: space - Predicted class: space PASS!
Actual class: C - Predicted class: C PASS!
Actual class: T - Predicted class: T PASS!
Actual class: S - Predicted class: S PASS!
Actual class: X - Predicted class: X PASS!
Actual class: J - Predicted class: J PASS!
Actual class: Q - Predicted class: Q PASS!
Actual class: G - Predicted class: G PASS!
Actual class: I - Predicted class: I PASS!
====================
Test accuracy (%): 92.85714285714286
Test error rate (%): 7.142857142857142
Number of misclassified classes: 2
Number of correctly classified classes 26
```

➢ 6.1.2 Creating the Web-service and hosting the Model:

➔ 6.1.2.1- Hosting on DigitalOcean:

- We used DigitalOcean as the hosting server for our model; this provides a virtual machine with high computational power and great availability to host our model on. The server exposes 2000 as the port number for other systems and apps to communicate with in order to use the model's detection functionality. Thereby, creating a web-service that has an API that systems can send post requests with the image of the sign they want to detect, the request will be received by the server and processed to crop out just the green box containing the user's hand to then be passed to the model and take the resulting letter to be a response back to the system that sent the request.

➔ 6.1.2.2- NGINX:

- This open-source software was used to route the requests coming to the server from port 2000 to port 80 for security measures. It also provides SSL (Secure Socket Layer) which converts the request from HTTP to (secured) HTTPS, thereby guaranteeing a secure connection to the server which is a tremendously crucial non-functional requirement.

➔ 6.1.2.3- Docker:

- This is a virtualization service that allows the application of the containerization concept, as it wraps the model hosted on the server inside an isolated container along with its needed library packages and its configuration files. This provides the web-service and our model the benefit of separability of functionalities and increases the modifiability; because, when we would like to add another functionality, we can simply wrap it in a different container, without affecting the previous one. Docker also ensures that communication between containers can easily be done through defined secure channels.
- The containerisation concept that's applied also provides concurrency of handling requests by the server, as the whole application or system can be hosted on a server that exposes a specified port for it, but the containers inside the system has each feature or functionality separated, so it will be directly sent to its specific container while serving other requests for other functionalities at the same time.

➢ 6.1.3- Talking Signs Mobile Application:

➔ 6.1.3.1- Purpose of Development:

- The mobile application was developed to emphasize our point of trying to deliver the most value to the end-user, this was already done when we created the web-service that any system or application can communicate with easily to get the benefits of the classifier model. However, we went to the extent of developing a mobile application that shows how this communication can happen and make it more available for the users to start using the sign detection functionality in their daily life through this application.

➔ 6.1.3.2- React Native Framework:

- The React Native Framework was chosen to develop the application as it's a cross platform framework that allows for deploying on both Android and IOS mobile phones, so it gives us more exposure.
- The application opens to a back camera that can be flipped using the flip button. Inside the camera component there is a capture button that is supposed to be clicked to capture the image of the user performing a specific sign. Below the camera component, there is a text area that translated letters should appear on, along with other buttons that facilitate the use of the application like the refresh button that clears the whole text area, the delete button that can be used to delete the last letter shown on the screen, and the sound button that allows the user to hear a voiced translation of the text shown. The application also offers a manual button on its NavBar to display an image of all the signs that can be performed along with their corresponding letters; so that users can easily learn sign language while using the app.

➔ 6.1.3.3- How the application works:

- The user opens the application and starts performing signs in the green box designed for his hand to be in. The user then presses the capture button to snap an image of the sign he performed. The image captured will be the whole image of the user, so the application will start to crop out the green box that has the user's hand and sends a post request containing this image to the web-service. The server will receive the request and will pass the frame to the classifier model to start its detection. The model will return the detected letter and the server will respond back to the application that's waiting for the response since it sent the request with the translated letter. The application will then check if the response it got is delete or space or nothing to handle each condition differently. If space is returned then the app will print out space instead of the word space itself. If del is returned, then the application will remove the last letter shown without showing the del word. If nothing is returned, then the app will print an alert message for the user to try again. If the response was none of the previous conditions, then it is a simple letter that the application will print on the screen thereby concatenating it with the rest of the text.

➔ 6.1.3.4- Snippet from the Implementation Code of the Mobile Application:

- **Camera Component:**

```
import React, { useState, useEffect, useRef } from 'react';
import { StyleSheet, Text, View, TouchableWithoutFeedback, ToastAndroid, Dimensions, Platform } from 'react-native';
import { Camera } from 'expo-camera';
import * as ImageManipulator from 'expo-image-manipulator'
import MaterialCommunityIcons from "react-native-vector-icons/MaterialCommunityIcons";
import axios from 'axios'


const CameraComponent = (props) => {
  const [hasPermission, setHasPermission] = useState(null);
  const [type, setType] = useState(Camera.Constants.Type.back);
  const [snapVisible, setVisible] = useState(true);
  const camRef = useRef(null);
  const frameSquare = Dimensions.get('window').width * 0.67;

  useEffect(() => {
    (async () => {
      const { status } = await Camera.requestPermissionsAsync();
      setHasPermission(status === 'granted');
    })();
  }, []);

  if (hasPermission === null) {
    return <View />;
  }
  if (hasPermission === false) {
    return <Text>No access to camera</Text>;
  }

  const sendFrame = async () => {
    setVisible(false);
    var img = await camRef.current.takePictureAsync({ quality: 1 });
    //ToastAndroid.show(img.uri, ToastAndroid.SHORT);
    var width = img.width;
    var square = width * 0.67;
    //console.log(type, Camera.Constants.Type);
    if (type === 0) {
      var manipResult = await ImageManipulator.manipulateAsync(
```

```
if (type === 0) {
  var manipResult = await ImageManipulator.manipulateAsync(
    img.uri,
    [{
      crop: {
        originX: Math.floor(width * 0.33),
        originY: 0,
        width: Math.floor(square),
        height: Math.floor(square)
      }
    },
    { resize: { width: 200, height: 200 } }]
  );
} else if (type === 1) {
  manipResult = await ImageManipulator.manipulateAsync(
    img.uri,
    [{
      crop: {
        originX: 0,
        originY: 0,
        width: Math.floor(square),
        height: Math.floor(square)
      }
    },
    { resize: { width: 200, height: 200 } },
    { rotate: 180 }, { flip: ImageManipulator.FlipType.Vertical }]
  );
}
img = manipResult
//console.log(img.width, img.height);

try {
  if (Platform.OS === 'android') {
    //ToastAndroid.show("Processing data", ToastAndroid.SHORT);
  }
  let uri = img.uri;
  let apiUrl = 'https://api.talkingsigns.cf/uploader';
```

```
    formData.append('file', {
      uri,
      name: `photo.jpg`,
      type: `image/jpg`,
    });

    let result = await axios.post(apiUrl, formData, {
      headers: {
        Accept: 'application/json',
        'Content-Type': 'multipart/form-data'
      }
    })
    props.onSubmit(result.data);

    if (Platform.OS === 'android') {
      //ToastAndroid.show("Api response: " + result.data, ToastAndroid.SHORT);
    }
  }
  catch (e) {
    console.log(e)
    if (Platform.OS === 'android') {
      ToastAndroid.show("image was not sent", ToastAndroid.SHORT);
    }
  }
  setVisible(true);
}

return (
  <View style={styles.container}>
    <Camera style={styles.camera} type={type}
      ref={camRef}
    >
      <View
        pointerEvents={"none"}
        style={{
          borderWidth: 4,
          borderColor: 'yellow',
```

```
        }} />
      <View style={styles.buttonContainer}>

        <TouchableWithoutFeedback
          style={styles.switch}
          onPress={() => {
            setType(
              type === Camera.Constants.Type.back
                ? Camera.Constants.Type.front
                : Camera.Constants.Type.back
            );
          }}>

          <MaterialCommunityIcons
            style={styles.switch}
            name="camera-switch"
            color={"#9c1937"}
            size={30}
          />

        </TouchableWithoutFeedback>

        {snapVisible ?
          <TouchableWithoutFeedback style={styles.snap}
            onPress={() => { sendFrame() }}
          >
            <MaterialCommunityIcons
              style={styles.snap}
              name="circle"
              color={"#9c1937"}
              size={50}
            />
          </TouchableWithoutFeedback>
          : <View />}
      </View>
    </Camera>
  </View>
```

(GitHub Link: https://github.com/YoussefAhmed99/talking-signs)

❖ 6.2- Testing:

➢ 6.2.1- Testing the Mobile Application:

Test Scenario	The user wants to display the manual of the signs.		
Test Case ID	MID1		
Test Priority	High		
Preconditions	None		
Test Steps	**Expected Output**	**Actual Output**	**Test Results**
1-User opens the application. 2-User clicks show manual button.	System switches the home screen and returns an image of all signs with corresponding letters on the screen.	As Expected	Passed

Test Scenario	The user wants to flip the camera.		
Test Case ID	MID2		
Test Priority	High		
Preconditions	None		
Test Steps	**Expected Output**	**Actual Output**	**Test Results**
1-User opens the application. 2-the application opens by default on back camera. 3-The user clicks on the flip icon to flip to front camera.	-The camera flips successfully.	-As expected	Passed

Test Scenario	The user wants to clear the text area.		
Test Case ID	MID3		
Test Priority	High		
Preconditions	User should have performed at least one sign and corresponding letter is shown on screen.		
Test Steps	**Expected Output**	**Actual Output**	**Test Results**
1-the user clicks on the refresh icon to delete the text-area.	-The text-area is deleted successfully.	-As expected	Passed

Test Scenario	The user wants to delete the last character shown in the text area.		
Test Case ID	MID4		
Test Priority	High		
Preconditions	User should have performed at least one sign and corresponding letter is shown on screen.		
Test Steps	**Expected Output**	**Actual Output**	**Test Results**
1-User clicks on delete button to remove last letter in text-area.	-The letter is removed successfully.	As expected	Passed

Test Scenario	The user wants to hear an audio translation of the text shown on the text area.		
Test Case ID	MID5		
Test Priority	High		
Preconditions	User should have performed at least one sign and corresponding letter is shown on screen.		
Test Steps	**Expected Output**	**Actual Output**	**Test Results**
1-The user clicks on the sound icon to hear a voiced translation of the text shown on the text-area.	- The application returns a voiced translation of the text-area.	As expected	Passed

Test Scenario	The user wants to capture the image of him performing a sign to be translated.		
Test Case ID	MID6		
Test Priority	High		
Preconditions	None		
Test Steps	**Expected Output**	**Actual Output**	**Test Results**
1-The user clicks on the capture button to take a photo of the performed sign.	-The system disables the visibility of the capture button once a user clicks on it to prevent clicking on it several times without processing the already captured frame. -The system enables the visibility of the capture button once processing is finished and response is received.	As expected	Passed

➢ 6.2.2- Testing the API:

➔ Testing Code:

```
import os
import requests

successTrials=0
numberoftrials=0

def checkdir(d):
    global numberoftrials
    global successTrials
    for file in os.listdir(d):
        url = 'https://api.talkingsigns.cf/uploader'
        filepath =os.path.join(d, file)
        classA = d.split("-")[0]
        classA = classA.split("\\")[1]
        files = {'file': open(filepath, 'rb').read()}
        response = requests.post(url, files=files)
        try:
            data = response.text
            numberoftrials = 1 + numberoftrials
            data= data.upper()
            classA=classA.upper()
            if(data.rstrip() == classA.rstrip()):
                print("Actual Class:", classA, "- Predicted Class:", data, "- PASS!")
                print("----------------------------------------------")
                successTrials= successTrials + 1
            else:
                print("Actual Class:", classA, "- Predicted Class:", data, "- FAIL!")
                print("----------------------------------------------")
        except:
            print("api testing failed")

def listdirs(rootdir):
    for file in os.listdir(rootdir):
        d = os.path.join(rootdir, file)
        if os.path.isdir(d):
            checkdir(d)

rootdir = 'Classes'
listdirs(rootdir)

print("Test Accuracy=",(successTrials/numberoftrials)*100, "%")
```

➔ **Testing Output:**

Actual Class: nothing
Predicted Class: nothing

Actual Class: space
Predicted Class: space

Actual Class: del
Predicted Class: del

Actual Class: A
Predicted Class: A

Actual Class: B
Predicted Class: B

Actual Class: C
Predicted Class: C

Actual Class: D
Predicted Class: D

Actual Class: E
Predicted Class: E

Actual Class: F
Predicted Class: F

Actual Class: G
Predicted Class: Y

Actual Class: H
Predicted Class: H

Actual Class: I
Predicted Class: I

Actual Class: J
Predicted Class: Y

Actual Class: K
Predicted Class: K

Actual Class: L
Predicted Class: L

Actual Class: M
Predicted Class: M

Actual Class: N
Predicted Class: N

Actual Class: O
Predicted Class: O

Actual Class: P
Predicted Class: P

Actual Class: Q
Predicted Class: Q

Actual Class: R
Predicted Class: R

Actual Class: S
Predicted Class: A

Actual Class: T
Predicted Class: L

Actual Class: U
Predicted Class: U

Actual Class: V
Predicted Class: V

Actual Class: W
Predicted Class: W

Actual Class: X
Predicted Class: X

Actual Class: Y
Predicted Class: Y

Actual Class: Z
Predicted Class: Z

Calculating Accuracy:

Actual Class: A - Predicted Class: A - PASS!

Actual Class: A - Predicted Class: A - PASS!

Actual Class: A - Predicted Class: A - PASS!

Actual Class: A - Predicted Class: A - PASS!

Actual Class: B - Predicted Class: B - PASS!

Actual Class: B - Predicted Class: B - PASS!

Actual Class: B - Predicted Class: B - PASS!

Actual Class: B - Predicted Class: B - PASS!

Actual Class: B - Predicted Class: B - PASS!

Actual Class: B - Predicted Class: B - PASS!

Actual Class: C - Predicted Class: C - PASS!

Actual Class: C - Predicted Class: C - PASS!

Actual Class: C - Predicted Class: C - PASS!

Actual Class: C - Predicted Class: C - PASS!

Actual Class: D - Predicted Class: D - PASS!

Actual Class: D - Predicted Class: D - PASS!

Actual Class: D - Predicted Class: D - PASS!

Actual Class: D - Predicted Class: D - PASS!

Actual Class: DEL - Predicted Class: DEL - PASS!

Actual Class: DEL - Predicted Class: DEL - PASS!

Actual Class: DEL - Predicted Class: DEL - PASS!

Actual Class: DEL - Predicted Class: DEL - PASS!

Actual Class: E - Predicted Class: E - PASS!

Actual Class: E - Predicted Class: E - PASS!

Actual Class: E - Predicted Class: E - PASS!

Actual Class: E - Predicted Class: E - PASS!

Actual Class: E - Predicted Class: E - PASS!

Actual Class: E - Predicted Class: E - PASS!

Actual Class: E - Predicted Class: E - PASS!

Actual Class: E - Predicted Class: E - PASS!

Actual Class: F - Predicted Class: F - PASS!

Actual Class: F - Predicted Class: F - PASS!

Actual Class: F - Predicted Class: F - PASS!

Actual Class: F - Predicted Class: F - PASS!

Actual Class: G - Predicted Class: G - PASS!

Actual Class: G - Predicted Class: G - PASS!

Actual Class: G - Predicted Class: G - PASS!

Actual Class: G - Predicted Class: Y - FAIL!

Actual Class: H - Predicted Class: H - PASS!

Actual Class: H - Predicted Class: H - PASS!

Actual Class: H - Predicted Class: H - PASS!

Actual Class: H - Predicted Class: H - PASS!

Actual Class: I - Predicted Class: I - PASS!

Actual Class: I - Predicted Class: I - PASS!

Actual Class: I - Predicted Class: I - PASS!

Actual Class: I - Predicted Class: I - PASS!

Actual Class: J - Predicted Class: J - PASS!

Actual Class: J - Predicted Class: J - PASS!

Actual Class: J - Predicted Class: J - PASS!

Actual Class: J - Predicted Class: Y - FAIL!

Actual Class: J - Predicted Class: J - PASS!

Actual Class: J - Predicted Class: J - PASS!

Actual Class: K - Predicted Class: K - PASS!

Actual Class: K - Predicted Class: K - PASS!

Actual Class: K - Predicted Class: K - PASS!

Actual Class: K - Predicted Class: K - PASS!

Actual Class: L - Predicted Class: L - PASS!

Actual Class: L - Predicted Class: L - PASS!

Actual Class: L - Predicted Class: L - PASS!

Actual Class: L - Predicted Class: L - PASS!

Actual Class: M - Predicted Class: M - PASS!

Actual Class: M - Predicted Class: M - PASS!

Actual Class: M - Predicted Class: M - PASS!

Actual Class: M - Predicted Class: M - PASS!

Actual Class: N - Predicted Class: N - PASS!

Actual Class: N - Predicted Class: N - PASS!

Actual Class: N - Predicted Class: N - PASS!

Actual Class: N - Predicted Class: N - PASS!

Actual Class: NOTHING - Predicted Class: NOTHING - PASS!

Actual Class: NOTHING - Predicted Class: NOTHING - PASS!

Actual Class: NOTHING - Predicted Class: NOTHING - PASS!

Actual Class: NOTHING - Predicted Class: NOTHING - PASS!

Actual Class: NOTHING - Predicted Class: NOTHING - PASS!

Actual Class: O - Predicted Class: O - PASS!

Actual Class: O - Predicted Class: O - PASS!

Actual Class: O - Predicted Class: O - PASS!

Actual Class: O - Predicted Class: O - PASS!

Actual Class: P - Predicted Class: P - PASS!

Actual Class: P - Predicted Class: P - PASS!

Actual Class: P - Predicted Class: P - PASS!

Actual Class: P - Predicted Class: P - PASS!

Actual Class: Q - Predicted Class: Q - PASS!

Actual Class: Q - Predicted Class: Q - PASS!

Actual Class: Q - Predicted Class: Q - PASS!

Actual Class: Q - Predicted Class: Q - PASS!

Actual Class: R - Predicted Class: R - PASS!

Actual Class: R - Predicted Class: R - PASS!

Actual Class: R - Predicted Class: R - PASS!

Actual Class: R - Predicted Class: R - PASS!

Actual Class: S - Predicted Class: S - PASS!

Actual Class: S - Predicted Class: S - PASS!

Actual Class: S - Predicted Class: A - FAIL!

Actual Class: S - Predicted Class: S - PASS!

Actual Class: SPACE - Predicted Class: SPACE - PASS!

Actual Class: SPACE - Predicted Class: SPACE - PASS!

Actual Class: SPACE - Predicted Class: SPACE - PASS!

Actual Class: SPACE - Predicted Class: SPACE - PASS!

Actual Class: T - Predicted Class: T - PASS!

Actual Class: T - Predicted Class: L - FAIL!

Actual Class: T - Predicted Class: T - PASS!

Actual Class: T - Predicted Class: T - PASS!

Actual Class: U - Predicted Class: U - PASS!

Actual Class: U - Predicted Class: X - FAIL!

Actual Class: U - Predicted Class: U - PASS!

Actual Class: U - Predicted Class: U - PASS!

Actual Class: V - Predicted Class: V - PASS!

Actual Class: V - Predicted Class: W - FAIL!

Actual Class: V - Predicted Class: V - PASS!

Actual Class: V - Predicted Class: V - PASS!

Actual Class: W - Predicted Class: W - PASS!

Actual Class: W - Predicted Class: W - PASS!

Actual Class: W - Predicted Class: W - PASS!

Actual Class: W - Predicted Class: W - PASS!

Actual Class: X - Predicted Class: X - PASS!

Actual Class: X - Predicted Class: X - PASS!

Actual Class: X - Predicted Class: X - PASS!

Actual Class: X - Predicted Class: X - PASS!

Actual Class: Y - Predicted Class: Y - PASS!

Actual Class: Y - Predicted Class: Y - PASS!

Actual Class: Y - Predicted Class: Y - PASS!

Actual Class: Y - Predicted Class: Y - PASS!

Actual Class: Z - Predicted Class: Z - PASS!

Actual Class: Z - Predicted Class: Z - PASS!

Actual Class: Z - Predicted Class: Z - PASS!

Actual Class: Z - Predicted Class: Z - PASS!

Test Accuracy (%): 95.19999999999999
Number of Trials: 125
Number of Correctly Classified Classes: 119

➢ 6.2.3- System Testing:

➔ 6.2.3.1- Functional Testing:

Test Scenario	The user wants to translate the sign he performed.		
Test Case ID	FID1		
Test Priority	High		
Preconditions	The server is up and running.		
Test Steps	**Expected Output**	**Actual Output**	**Test Result**
1- The user opens the application. 2- The user performs a sign. 3- The user presses on the capture button to capture a photo of the performed sign.	- The application captures the frame and crops the green box that exists on the frame; such that only the cropped part (just the user's hand) is the main frame. - The application then sends the cropped frame through a post request to the server hosting the classifier model. - The frame is received by the server and is pre-processed to an 200x200 image and is then passed to the classifier. - The classifier model predicts the passed image and the detected letter is then sent back by the server as a response to the application's request. - The application receives the response from the server and displays the letter on the screen's text area for the user to see.	As expected	Passed

Test Scenario	The user wants to translate the sign he performed.		
Test Case ID	FID2		
Test Priority	High		
Preconditions	The server is down.		
Test Steps	**Expected Output**	**Actual Output**	**Test Results**
1- The user opens the application. 2- The user performs a sign. 3- The user presses on the capture button to capture a photo of the performed sign.	- The application captures the frame and crops the green box that exists on the frame; such that only the cropped part (just the user's hand) is the main frame. - The application then sends the cropped frame through a post request to the server hosting the classifier model. - The frame is not received by the server as the server is not available. - The application returns a message on the screen for the user to try again later as the image was not sent.	As expected	Passed

Test Scenario	The user wants to translate the space sign he performed.		
Test Case ID	FID3		
Test Priority	High		
Preconditions	None.		
Test Steps	**Expected Output**	**Actual Output**	**Test Results**
1- The user opens the application. 2- The user performs the (space) sign. 3- The user presses on the capture button to capture a photo of the performed sign.	- The application captures the frame and crops the green box that exists on the frame; such that only the cropped part (just the user's hand) is the main frame. - The application then sends the cropped frame through a post request to the server hosting the classifier model. - The frame is received by the server and is pre-processed to an 200x200 image and is then passed to the classifier. - The classifier model predicts the passed image correctly and (space) word is then sent back by the server as a response to the application's request. - The application receives the response from the server and (if-space-condition) will be true, so a space will be left on the screen without displaying the word (space) itself.	As expected	Passed

Test Scenario	The user wants to delete the last letter shown in the text area by performing the delete sign.		
Test Case ID	FID4		
Test Priority	High		
Preconditions	The user should have performed at least one sign and the corresponding letter is shown on screen.		
Test Steps	**Expected Output**	**Actual Output**	**Test Results**
1- The user opens the application. 2- The user performs the (delete) sign. 3- The user presses on the capture button to capture a photo of the performed sign.	- The application captures the frame and crops the green box that exists on the frame; such that only the cropped part (just the user's hand) is the main frame. - The application then sends the cropped frame through a post request to the server hosting the classifier model. - The frame is received by the server and is pre-processed to an 200x200 image and is then passed to the classifier. - The classifier model predicts the passed image correctly and (del) word is then sent back by the server as a response to the application's request. - The application receives the response from the server and (if-dee-condition) will be true, so the app will remove the last letter shown on the screen without displaying the word (del) itself.	As expected	Passed

Test Scenario	The user performs a sign that could not be detected by the model. (nothing)		
Test Case ID	FID5		
Test Priority	High		
Preconditions	None		
Test Steps	**Expected Output**	**Actual Output**	**Test Results**
1- The user opens the application. 2- The user performs a sign. 3- The user presses on the capture button to capture a photo of the performed sign.	- The application captures the frame and crops the green box that exists on the frame; such that only the cropped part (just the user's hand) is the main frame. - The application then sends the cropped frame through a post request to the server hosting the classifier model. - The frame is received by the server and is pre-processed to a 200x200 image and is then passed to the classifier. - The classifier model tries to predict the passed image but detects it as (nothing), so (nothing) word is then sent back by the server as a response to the application's request. - The application receives the response from the server and (if-nothing-condition) will be true, so the app will display a try again message to the user without displaying the word (nothing) itself.	As expected	Passed

➔ 6.2.3.2- Non-Functional Testing:

Test Scenario	The user opens the application and grants access for the camera component.		
Test Case ID	NID1 (Security)		
Test Priority	High		
Preconditions	None		
Test Steps	**Expected Output**	**Actual Output**	**Test Results**
1- The user opens the application.	- The application asks the user for permission to access the camera. - The user grants permission and allows access to the camera. - The application opens the camera component and the text area is shown below it.	As expected	Passed

Test Scenario	The user opens the application and denies access for the camera component.		
Test Case ID	NID2 (Security)		
Test Priority	High		
Preconditions	None		
Test Steps	**Expected Output**	**Actual Output**	**Test Results**
1- The user opens the application.	- The application asks the user for permission to access the camera. - The user denies permission and doesn't allow access to the camera. - The application displays just the text area view and the manual view with no camera component shown.	As expected	Passed

Test Scenario	The user performs a sign and the response time is monitored.		
Test Case ID	NID3 (Performance)		
Test Priority	High		
Preconditions	None		
Test Steps	**Expected Output**	**Actual Output**	**Test Results**
1- The user opens the application. 2- The user performs a sign. 3- The user presses on the capture button to capture a photo of the performed sign.	- The response is received from the API and shown on the screen in a time that doesn't exceed 1 second.	Time taken to receive a response is 278 milliseconds.	Passed

- **References:**

[1] Dardas N, Chen Q, Georganas ND, Petriu EM (2010) Hand gesture recognition using bag-of-features and multi-class support vector machine. In: Haptic audio-visual environments and games (HAVE), 2010 IEEE international symposium, IEEE, pp 1–5

[2] Lionnie R, Timotius IK, Setyawan I (2012) Performance comparison of several pre-processing methods in a hand gesture recognition system based on nearest neighbor for different background conditions. J ICT Res Appl 6:183–194

[3] Zaki MM, Shaheen SI (2011) Sign language recognition using a combination of new vision based features. Pattern Recognit Lett 32:572–577

[4] Taskiran, M., Killioglu, M., & Kahraman, N. (2018). A Real-Time System for Recognition of American Sign Language by using Deep Learning.2018 41st International Conference on Telecommunications and Signal Processing (TSP).

[5] Cao Dong, Ming C. Leu, Zhaozheng Yin; Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops, 2015, pp. 44-52

[6] M. M. Islam, S. Siddiqua, J. Afnan, "Real time Hand Gesture Recognition using different algorithms based on American Sign Language," in Proc. IEEE International Conference on Imaging, Vision Pattern Recognition (icIVPR), Dhaka, Bangladesh, 2017, pp. 1–6.

[7] Pan TY, Lo LY, Yeh CW, Li JW, Liu HT, Hu MC (2016) Realtime sign language recognition in complex background scene based on a hierarchical clustering classification method. In: Multimedia big data (BigMM), 2016 IEEE second international conference, IEEE, pp 64–67

[8] Gupta B, Shukla P, Mittal A (2016) K-nearest correlated neighbor classification for Indian sign language gesture recognition using feature fusion. In: 2016 international conference on computer communication and informatics (ICCCI), IEEE, pp 1–5