ASHESI

ASHESI UNIVERSITY COLLEGE

BUILDING A WEB-BASED CONTINOUS SIGN LANGUAGE TRANSLATION APPLICATION FOR THE DEAF AND HARD-OF-HEARING (DHH)

APPLIED PROJECT

B.Sc. Computer Science

Samuel Atule

2020

ASHESI UNIVERSITY COLLEGE

BUILDING A WEB-BASED CONTINOUS SIGN LANGUAGE TRANSLATION APPLICATION FOR THE DEATH AND HARD-OF-HEARING (DHH)

APPLIED PROJECT

Applied Project submitted to the Department of Computer Science, Ashesi University College in partial fulfilment of the requirements for the award of Bachelor of Science degree in Computer Science.

Samuel Atule

2020

DECLARATION

I hereby declare that this Applied Project is the result of my own original work and that no part of it has been presented for another degree in this university or elsewhere.

Candidate's Signature:

…………………………………………………………………………………………

Candidate's Name:

…………………………………………………………………………………………

Date:

…………………………………………………………………………………………

I hereby declare that preparation and presentation of this Applied Project were supervised in accordance with the guidelines on supervision of Applied Project laid down by Ashesi University.

Supervisor's Signature:

…………………………………………………………………………………………

Supervisor's Name:

…………………………………………………………………………………………

Date:

…………………………………………………………………………………………

Acknowledgements

Writing this Applied Project was harder than I thought, and more rewarding than I could have ever imagined. Thank you to Dad, the Mastercard Foundation, and Ashesi University, who took a chance on me, and have given me this transformative education. To the Ashesi community, colleagues, and friends whose warmth and acceptance made my learning experience a great one. To these colleagues: Mustapha Yusif Tidoo and Sabastien Dovonon Gbetonji and to my lecturer Dr. Ayorkor Korsah all of whom have been instrumental in my academic and career paths. To my amazing lecturer and friend, Dr. Esi Ansah, and to Florence Mawusi Ofori all of whom were always there anytime I went AWOL. Thank you for your love and support.

And lastly, to my supervisor, Dr. David Ebo Adjepon-Yamoah, whose encouragement, and academic advice helped me undertake this project. Thank you.

Abstract

Advancement in technology has brought about limitless possibilities. We have seen technology been applied to every aspect of our lives, from Medicine to the food we eat. This technology can also be applied to make life more comfortable for the Deaf and Hard-of-Hearing (DHH) individuals. Assistive technologies have been built for individuals with various forms of disabilities. DHH individuals have seen assistive technologies in the form of mechanical hardware, and cybernetics that enables translation of sign language to verbal communication in various languages. However, these technologies are often expensive and difficult to afford. In this project, an automatic sign language translation model is implemented to help people without hearing impairment to communicate better with DHH individuals in order to better provide services for them.

Table of Content

Chapter 1: Introduction

1.1 Background

The global commitment to reduce barriers for persons with disabilities, coupled with unparalleled advancement in technology, has motivated the development of many assistive technologies in recent times. One category of persons with disability which have been greatly disadvantaged is the Deaf and Hard-of-Hearing (DHH). Deaf people experience higher rates of unemployment and underemployment and earn lifetime wages that are between $356,000 and $609,000 less than their comparably educated normal-hearing counterparts [8]. [8] further observed that this substantial loss of earning power reflects the lack of understanding of the cultural and communication needs of deaf people. [1] duplicated the observation made in [8] among DHH patients and health workers in South Africa, Brazil, the United States, and Netherlands. [1] conducted a study examining the quality of communication between 26 General Practitioners (GPs) and 32 of their deaf patients. The authors found that only 13% of doctors and patients evaluated their GP-Patient communication as good against 39% of cases where the communication was rated moderate or bad. In the United Kingdom, 44% of deaf patients found the last contact with their GP or health center difficult or very difficult comparable to 17% from a general population survey, according to [1] Around 466 million people worldwide representing over 5% of the entire world population have disabling hearing loss, the prevalence of these cases is greatest in Sub-Sahara Africa, South Asia, and the Asia Pacific [10].

1.2 Related Works

The significance of this problem has necessitated specific solutions. In order to reduce the communication gap among DHH people, some authors have proposed and developed disability-friendly user interfaces to enable DHH personnel to consume information in different formats. [6] built a web-based sign language animation avatar that allowed DHH persons to easily access text-based web documents available for DHH persons who could only communicate in sign language. [9] also implemented a speed and pausing animation model for the American Sign Language system. However, other authors such as [12], [7], and [11] all used physical hardware such as sensor gloves or cybernetics to ensure the continuous real-time sign language recognition and interpretation in a workplace context. However, these authors observed that the cost of hardware is very expensive and requires that either the DHH person or the normal person wear them before communication could ensue. [4] employed a combination of statistical probability and simple Recurrent Networks (SRN) and developed a continuous Chinese Sign Language (CSL) system to transcribe Chinese sign language into text or speech. Realizing the limitation of SRN since it often uses hand-crafted features to describe sign language motion, [5] instead employed 3D Convolutional Neural Network to implement an automatic sign language recognition model. However, [5] noted the expensive nature of computing power needed to build such a model given the high computation needed to model hand motions and the prodigious amount of data.

1.3 Motivation and Problem Significance

Ashesi University is an educational institution with the mission to educate a new generation of ethical and entrepreneurial leaders in Africa. The institution prides itself on the beautiful and accessible nature of its building for people with disabilities. [13] studied accessible built environments among universities in Ghana and ranks Ashesi University as the second best in the Country. However, the school does not currently admit deaf students and those with Hard-of-Hearing challenges. Currently, the institution delivers teaching through traditional audiovisual means such as Projectors and Whiteboards. This, however, does not provide the facility to teach students with DHH challenges.

Furthermore, another case can be made for communication between Practitioners and DHH individuals during access to public services, where facilities are not usually provided for DHH individuals. As many as 70% of sign language users have said they have been left without interpreters during their time in accident and emergency departments throughout 2010 [3]. Apart from the difficulty in communication that is making it difficult to access health care services as illustrated in the context of this paper above, the study [2] on the issues facing deaf people when accessing public services concluded that Accessing an Interpreter, Accessibility of written English, Communication & Deaf awareness of Frontline Staff are some of the major challenges.

Project DHH seeks to provide a fast and reliable sign language translation that will enhance communication among DHH individuals and those without hearing impairment. This will serve to introduce accessibility facilities at Ashesi and other universities to provide tuition for students with DHH challenges, in the future, thereby creating a holistic effort to ensure that all manner of students can access quality education at all levels. Furthermore, this project will ensure that written English is readily accessible to DHH individuals, remove the need for interpreters at public institutions, and ensure that DHH individuals are not isolated within the context of our society. The proposed implementation is a mobile-based sign language translation application that can take communication in sign language and translate it into its equivalent written English. The reverse approach of translating English test into an avatar to ensure two-way communication will be implemented in a future iteration of this project.

Chapter 2: Requirements

2.1 Project Scope

Project DHH will be implemented as an Application Programming Interface (API), where it can be accessed by both a web-based environment and a mobile environment. For the purposes of this academic study, this project will implement the API and make it publicly accessible. The version one of this project will then build a user-oriented web-based application where users can access the functionalities of the API. In totality, this project will span over a Python API made publicly accessible, a web-based interface to access and control the functionalities of the API. Furthermore, and most importantly, the scope of the scenarios in which this application can be used to effectively facilitate sign language communication will largely depend on the nature of the data the machine learning model will be trained on. Since I do not have access to large amount of data, and the data will be manually collected, this project will be implemented to facilitate conversations occurring within the confines of the classroom building. Therefore, the scope of this project also extends to data collection. Considering the limited range of scenarios that this application can support as well as the novel architecture of the machine learning model that is implemented, Project DHH can be viewed as proof of concept. In this regard, this project will implement a data pipeline to make training of the model very easy in the future when more data is acquired. In totality Project DHH will thus have the following components:

I. Manual data collection limited to conversations within the classroom setting.
II. Deep Learning model implementation and training.
III. A 3D Recurrent Convolutional Neural Network(3DRCNN) and training.
IV. Building and deploying 3DRCNN as API.
V. Implementation of a web-based interface or a mobile application interface.
VI. Deploying the web-based application interface.

VII. Implementing a pipeline for continuous training and deployment as more data is made available.

As stated above, Project DHH can also be viewed as a prototype to evaluate the accuracy and efficiency of the model architecture employed in this project. As a result, in this system:

I. The mobile-based application interface will only be implemented in future iterations of the project.

II. The web-based application interface does not have a user-specific profile.

III. The system will be deployed on a lightweight server, and so the context of use will be highly localized for initial user testing.

2.2 Technical Product Description

Project DHH is a deep learning application with a web-based interface that translates sign language into English to facilitate communication between DHH individuals and those without hearing impairment.

Project DHH works by using a web-based application interface to access the user camera in order to take video clips of a user signing. This video data is then passed to the trained 3DRCNN deep learning model, which is implemented in the Python Keras framework and deployed as an API, through some endpoint. The Keras backend model predicts the English translation of the sign language communication in the video data and sends the prediction back to the web-based application interface to be displayed through another endpoint.

2.3 Project Objectives

The following objectives will be accomplished by the end of this academic project:

I. Employ a deep learning architecture that ensures faster training time and efficient use of computer memory.

II. Build a deep learning model that translates sign language to English within near real-time.

III. Implement and deploy a deep learning model as an API for public accessibility.

IV. Build a web-based application interface for users to access model functionalities.

V. Implement a data pipeline for continuous training and deployment of the deep learning model.

2.4 System Users

This system is primarily intended for two users: (i) Public Users (target users) (ii) Technical Users (periphery users).

2.4.1 Public Users

These are the primary users, and whose needs this system will broadly address. These are people with DHH challenges, and those without DHH challenges but who generally communicate with DHH individuals. They may include individuals but also public institutions that serve the needs of people with DHH challenges. These users are considered the early adopters and will form a majority of users who will test this system's functionalities for feasibility.

2.4.2 Technical Users

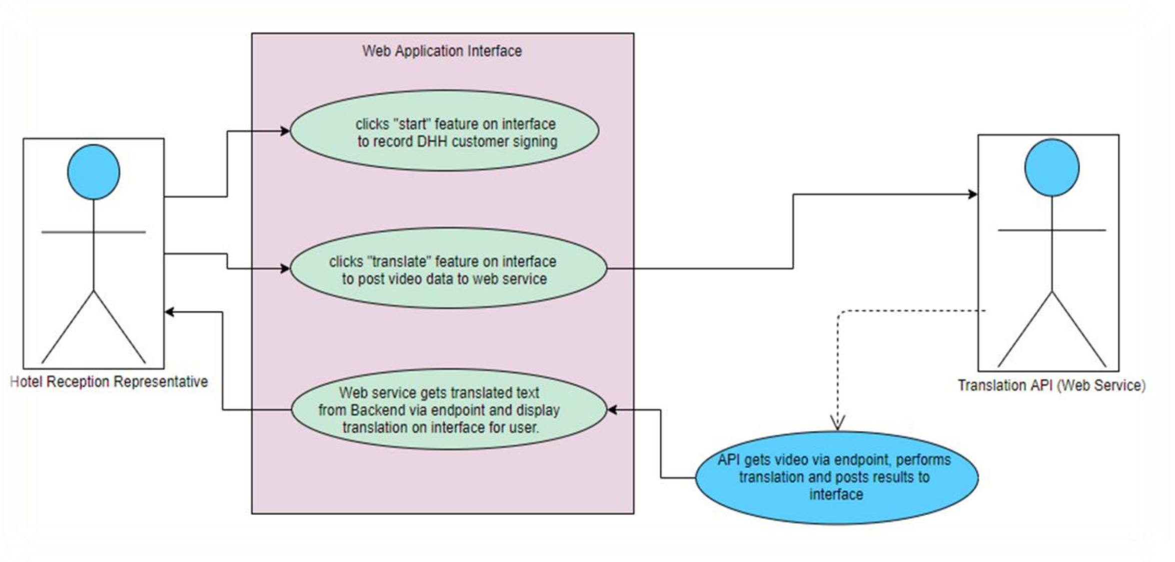
These are users who are only interested in accessing the underlying technologies of the system's backend API but not necessarily the entire system. They include other software developers who want to build other products around the functionalities of the system's backend API. These users form only a small base of this system's users.

2.5 Use Cases and Diagrams

2.5.1 Public User Use Case Scenario

An individual without DHH challenges wishes to communicate with another person with DHH challenges. So, she proceeds to lunch the web application interface either on her computer or using her mobile phone. The user wishing to understand the sign language communication of the person with the DHH challenges proceeds to click on the “start” feature on the web interface. This action opens the webcam of the computer or camera of the mobile phone, and the user can focus it on the DHH person who is signing. After the DHH is done communicating, the user can simply press the “translate” button on the web interface, and the equivalent translation of the sign language in English is immediately displayed below the camera footage on the web interface for the user’s perusal. These actions can be repeated anytime an individual decides to communicate with the DHH user.

2.5.2 Public User Use Case Diagram



Web Application Interface
clicks "start" feature on interface to record DHH customer signing
clicks "translate" feature on interface to post video data to web service
Web service gets translated text from Backend via endpoint and display translation on interface for user.
Hotel Reception Representative
Translation API (Web Service)
API gets video via endpoint, performs translation and posts results to interface

Figure 2.1: Public User Use Case Diagram

2.5.3 Technical User Use Case Scenario

A computer developer wishes to develop a mobile application that has the capability of translating sign language communication to the English language. Instead of implementing a translation engine from scratch, she decides to access the API documentation of the publicly hosted Sign Language translation backend of this system, and then build her mobile application interface to access this API directly. The documentation may also guide her on how to harvest more data, and to further train or fortune the functionalities of the backend API. Furthermore, a researcher may wish to study the accuracy or capabilities of Sign Language Recognition (SLR) models available publicly. This model can be directly downloaded, and the results reproduced with ease using the data pipeline of the model.

2.5.4 Technical User Use Case Diagram

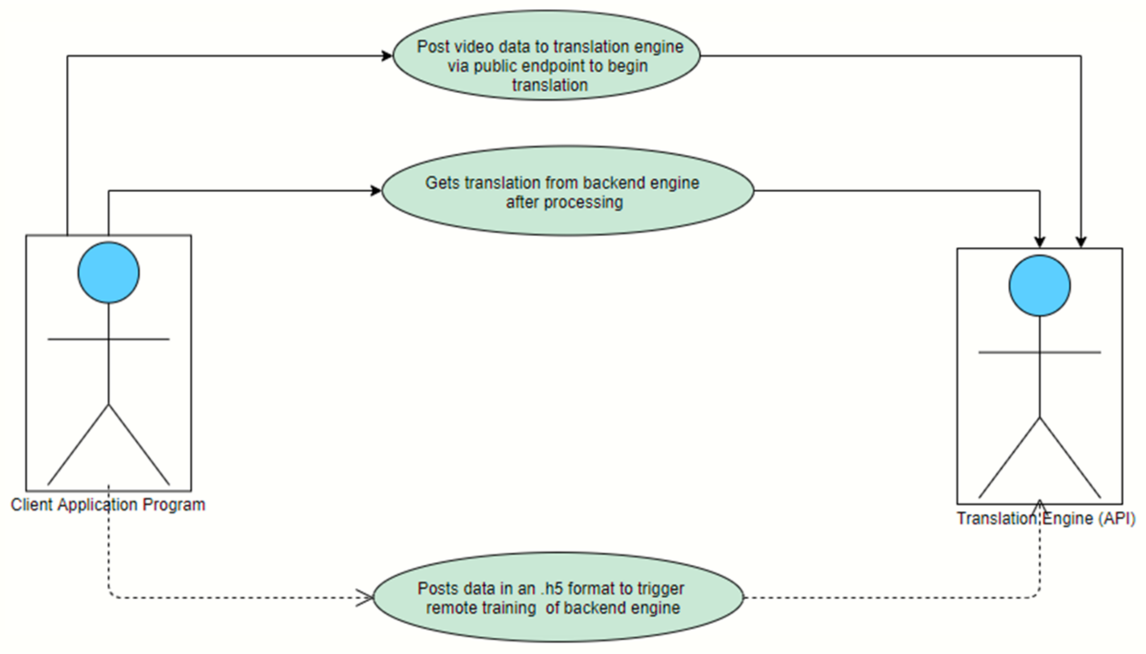

Post video data to translation engine via public endpoint to begin translation
Gets translation from backend engine after processing
Client Application Program
Translation Engine (API)
Posts data in an .h5 format to trigger remote training of backend engine

Figure 2.2: Technical User Use Case Diagram

2.6 Requirements

2.6.1 Procedure for Requirement Gathering and Analysis

Requirements were elicited through an intended literature review. For instance, as mentioned earlier, [5] implemented a 3D Convolutional neural network for automatic sign language recognition and noted the expensive nature of computing power needed to build such a model. Furthermore, the authors also noted the large amount of data needed to create such a model.

Moreover, conducting some ethnography research at the Mampong Deaf School, the observation made was that most of the pupils there did not have access to android phones; however, the school's IT laboratory was equipped with computers. This also informed the type of application interface that this project implemented. Finally, the author's domain knowledge from working with deep learning was also employed during the requirements gathering and analysis phase of this project.

2.6.2 Requirement Specification

2.6.2.1 Functional Requirements

2.6.2.1.1 Translation Model Requirements

This is the core component of the system and is responsible for the translation of sign language communication into English text. It is implemented as a deep learning model which takes in video data containing the sign language communication, and response with a translation of the communication into English for the user.

<u>Request/Response Sequence</u>

(i) The user lunches the application interface on the web or through their android phone

(ii) The user clicks on the “start” button on the interface, and then the user’s camera is accessed to start recording video of the DHH person communication.

(iii) The user clicks on the “translate” button, and the video recording is sent to the translation engine.

(iv) The translation engine takes as input the video stream and returns the English translation as the response.

(v) The response is displayed on the application interface for the user to see.

<u>User Requirements</u>

(i) The user needs the translation engine to perform near real-time prediction to ensure the conversation does not stall.

(ii) The user needs an accurate translation of sign language communication to prevent misinterpretation.

(iii) The user needs faster training time to ensure the model can be integrated into new systems.

<u>System Requirements</u>

(i) The translation engine must send a response reliably to facilitate integration.

(ii) The system must implement an appropriate mechanism to report feedback upon request completion.

2.6.2.1.2 Web Service (API) Requirements

This subcomponent ensures that other systems can access the functionalities of the Translation engine of this system. This is a web service implemented on top of the Translation engine with publicly available documentation of the various endpoints so that another user (technical user) can build a system with the Translation engine.

<u>Request/Response Cycle</u>

(i) The user accesses the API documentation of the web service available publicly online.

(ii) The user builds a system (interface) that accesses the “post” Uri of the web service and posts video data containing sign language communication to this URI of the web service.

(iii) The user retrieves a response that signal prediction was completed using a “get” Uri from the web service available in the documentation.

(iv) The user retrieves the web service’s response containing the translation of the sign language using another “get” Uri from the web service.

(v) The user’s system then consumes this response however they choose to.

User Requirements

(i) The user needs the API document available publicly in other to access the various functionalities of the web service.

(ii) The user needs the system to accept the request as well as respond with data in the specified format, as stated in the API documentation.

System Requirements

(i) The system must implement a “get” request endpoint to retrieve video data from the application interface when a user clicks the “translate” button.

(ii) The system must implement a “post” request to make the response available to the application interface after translation is complete.

(iii) The system must respond with a reliable data format.

2.6.2.1.3 Web Application Interface Requirements

This feature enables public users to be able to access the functionalities of the backend translation web service. This could be implemented as a web application interface or a mobile application interface. For the purpose of this study, a web-based application interface will be implemented.

Request/Response Sequence

(i) The user lunches the application interface on the web or through their android phone.

(ii) The user clicks on the “start” button on the interface, and then the user’s camera is accessed to start recording video of the DHH person communication.

(iii) The user clicks on the “translate” button, and the video recording is sent to the translation engine.

(iv) The translation engine takes as input the video stream and returns the English translation as the response.

(v) The response is displayed on the application interface for the user to see.

2.6.2.2 Non-Functional Requirements

Usability: The system should be designed for ease of use and learnability by users.

Simple Intuitive Interface: The system must have an interface that is simplified, easy to use, and minimizes the steps taken to accomplish a task.

Highly Diverse Dataset: The system must be trained with diverse data in order to translate user communication with high accuracy regardless of the origin of the user.

Reliability: The system must provide translation with high accuracy and should be able to handle errors such as incorrect data format to ensure that system is always available.

Light Weight Application Interface: The application must not consume a lot of resources so that it can run on several computing environments.

Near Real-Time Translation: The model must be lightweight to ensure faster response time in order to prevent users from waiting too long.

Privacy Protection: The system must secure user data to ensure that user conversations are secured from external attacks and data leakage.

Maintainability: The system must be easy to maintain or modify to ensure the scalability of the system.

2.7 Design and Implementation Constraints

As stated above in the scope of this project, scenarios in which this application can be used to effectively facilitate sign language communication will largely depend on the nature of the data the deep learning model will be trained on. Since I do not have access to a large amount of data, this project will be implemented to facilitate conversations occurring within the confines of the classroom building. As such, communications outside this scope may be less accurate.

Furthermore, this system is implemented and trained on Google's free tier GPU available on Google Collaboratory with only 12 Gigabytes of RAM. Therefore, computing resources will pose a significant challenge, which will increase training time and may decrease the number of design iterations that can be made within the limited time of this study. Moreover, the size of the deep learning architecture, which will be used to increase accuracy, may be affected since the size of the model depends on the number of computing resources available. Furthermore, Google Collaboratory also limits the amount of data that can be stored during model training, thereby decreasing the amount of data used to train the model.

Chapter 3: Architecture and Design

3.1 Design Specification

Project DHH system is created by implementing a deep learning architecture called Three-dimensional Recurrent Convolutional Neural Network (3DRCNN) using an encoder-decoder at the recurrent layer, a web service, and an application interface. The process that the video data containing the sign language communication goes through is shown in Figure 3.1.

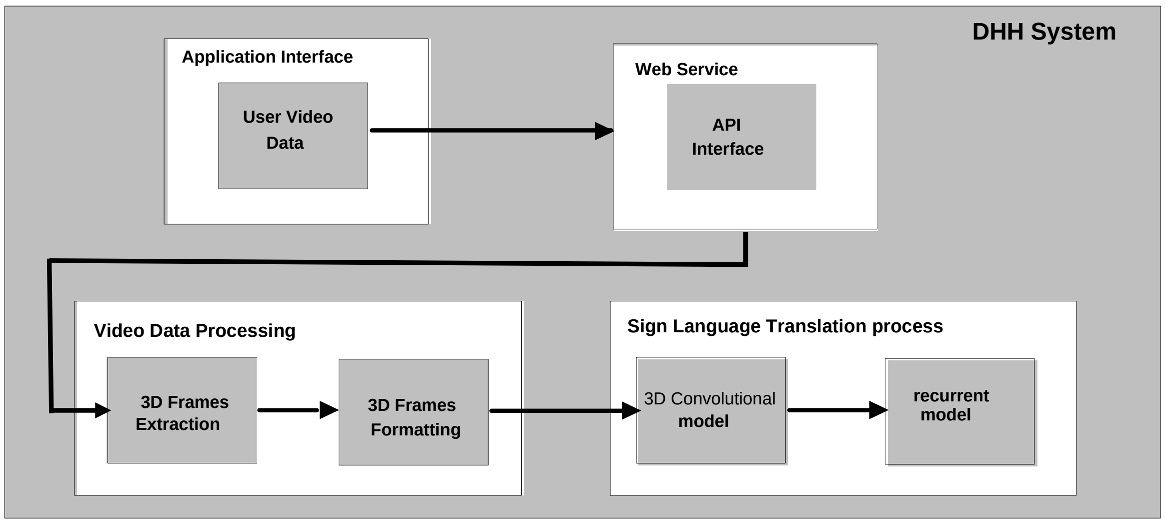


Figure 3.1: Flow chart diagram for the system

3.2 Module Decomposition

As illustrated in Figure 3.1 above, Project DHH is divided into four main modules; each one represents a high-level step in the principal system.

The modules are:

- Application Interface
- Web Service
- Data Processing Pipeline
- Translation Model

3.3 Architecture Description

The system generally employs the Model View Controller (MVC) in order to easily accommodate future changes to the entire system. The system implements each of the modules above separately, without introducing dependencies among the other components of the system. Furthermore, a web service architecture is employed at the backend to reduce critical system failures when incorporating more services in the future.

3.3.1 Module 1 — Application Interface

The system uses a [web or mobile] application interface to access the underlying functionalities of the translation engine through a web service. This interface is simplified to include two buttons to record video data and trigger the backend translation engine. A text field is created on the interface to display the response from the backend engine.

3.3.2 Module 2 — Web Service

The system employs a web service architecture to manage all components of the system. This means the web service coordinates requests and responses between the application interface(s) and the underlying translation model.

Another approach is to directly couple the translation engine together with interface either as a monolithic web application or a mobile application. The advantage of implementing this system as a monolithic application is that users will be able to access the functionalities of this application without the need for internet access. Furthermore, the monolithic architecture will increase processing time, leading to near Realtime translation. Although the monolithic architecture would be efficient in achieving the objectives of this project in terms of near Realtime translation, and faster computational time, there are significant challenges to employing this architecture, and the reason why this architecture was not implemented in this project.

TensorFlow (the machine learning framework that was used to implement the translation engine) provides the TensorFlow Lite used to deploy machine learning models directly on the mobile environment, it has yet to add features that allow for on-device training, or easy deployment on mobile environments. This means that the machine learning model cannot be trained on mobile devices, a lot of time is needed to deploy a complex model such as this system on to a mobile environment.

Therefore, the Web service architecture was implemented to circumvent the challenges stated above. The advantages are that the client device will not spend resources doing computation locally, which for most client devices, would likely take too long to complete translation. Furthermore, client application interfaces would be lightweight since the backend is hosted on a separate server. This means that a wide variety of client devices can install the system without worrying about hardware limitations. Moreover, a web service enforces uniformity. This means different computing environments will be able to access and extend the underlying functionalities of the translation engine.

Other advantages of Web Service Architecture are:

(i) Improves model performance since a single model is trained and utilized by all users rather than having an individual model for each user.

(ii) Model performance can also be tracked since a single model is used.

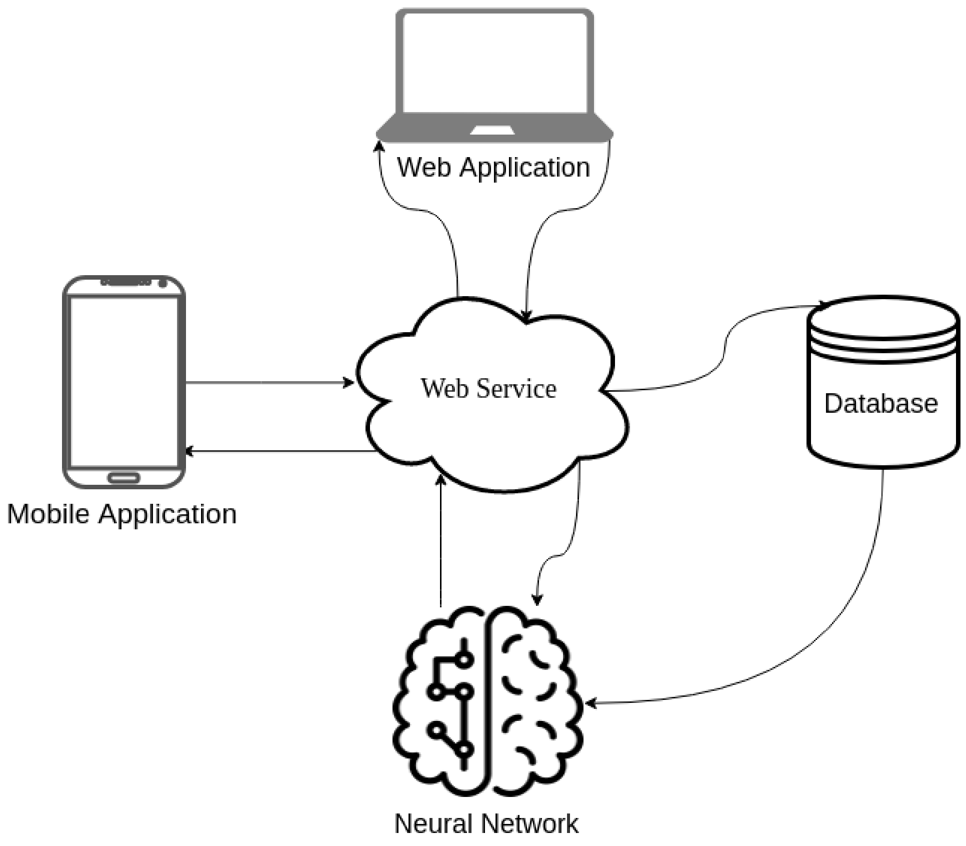


Figure 3.2: Schematic diagram of Web Service Architecture

3.3.3 Module 3 — Data Processing Pipeline

The system implements a data processing pipeline that processes the video data into a reliable format for the translation model. This module takes as input the video recording containing the sign language communication and returns an array of tensors in the format (40, 128, 128, 3) or (40, 224, 224, 3) to the convolutional neural network (CNN) of the translation model. To achieve this, the video recording is first extracted into colored frames, and each array converted into an array of pixels. The number of frames from each video recording may differ, and hence frames from each video are sampled into a fixed size of 40. Each of these 40 frames is then converted into an appropriate aspect ratio and then cropped into a shape of (224, 224, 3) or (128, 128, 3) depending on the CNN model that is used for feature extraction. At this point, the data is in the correct format to be passed to the translation model.

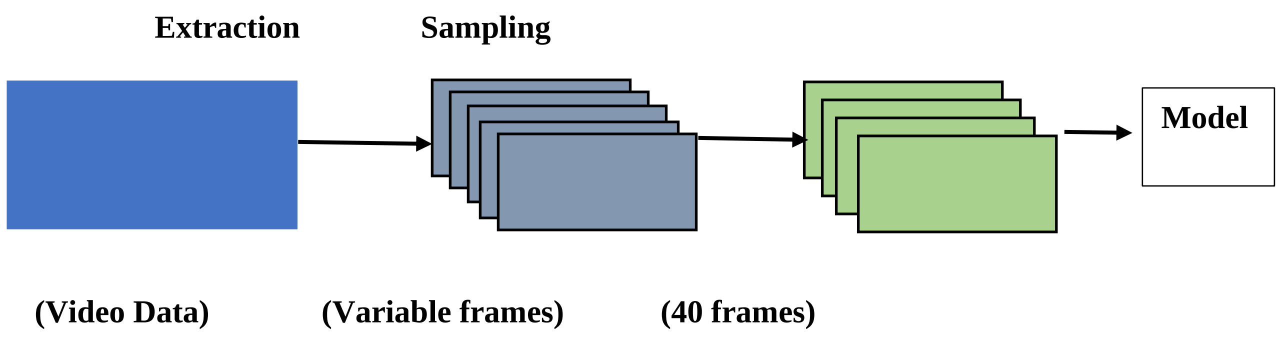


Figure 3.3: Schematic diagram of the Data Processing Pipeline

3.3.4 Module 4 — Translation Model

This is the core module of the system and implements a deep learning model that accepts an array from frames extracted from the data processing module and predicts the translation of the sign language. A 3-Dimensional Recurrent Convolutional Neural Network (3D-RCNN) architecture is employed in this model.

Considering the limited dataset available for this project and the high computational requirement of this deep learning model, the convolutional model was not implemented from scratch as this will led to overfitting resulting in low accuracy. Instead, transfer learning was employed using ReceptionV3 convolutional and the popular MobileNet model available in the Keras Machine Learning Framework. The output layers of these models are removed, leaving only the pooling layers in order to extract high-level spatial features of the frames. The extracted 40 frames of the video data are fed into the convolutional model in order to retrieve the spatial features of each frame. After the convolutional model, a tensor of shape (40, 1024) is returned and saved locally. The extracted spatial features saved locally are then passed to a recurrent model, which is implemented as an Encoder-Decoder model.

The encoder-decoder model is implemented and trained from scratch to exploit the temporal information from the 40 extracted features and predict translation based on this feature. This model is trained end-to-end using the extracted features from the convolutional model.

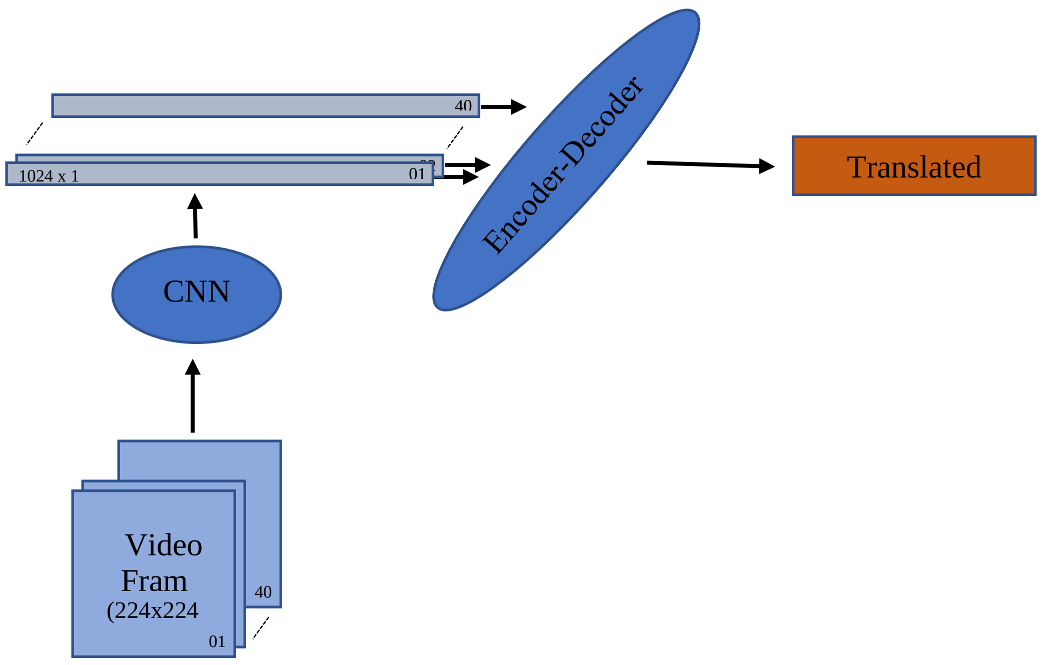


Figure 3.4: Schematic diagram of 3D-RCNN Model Architecture

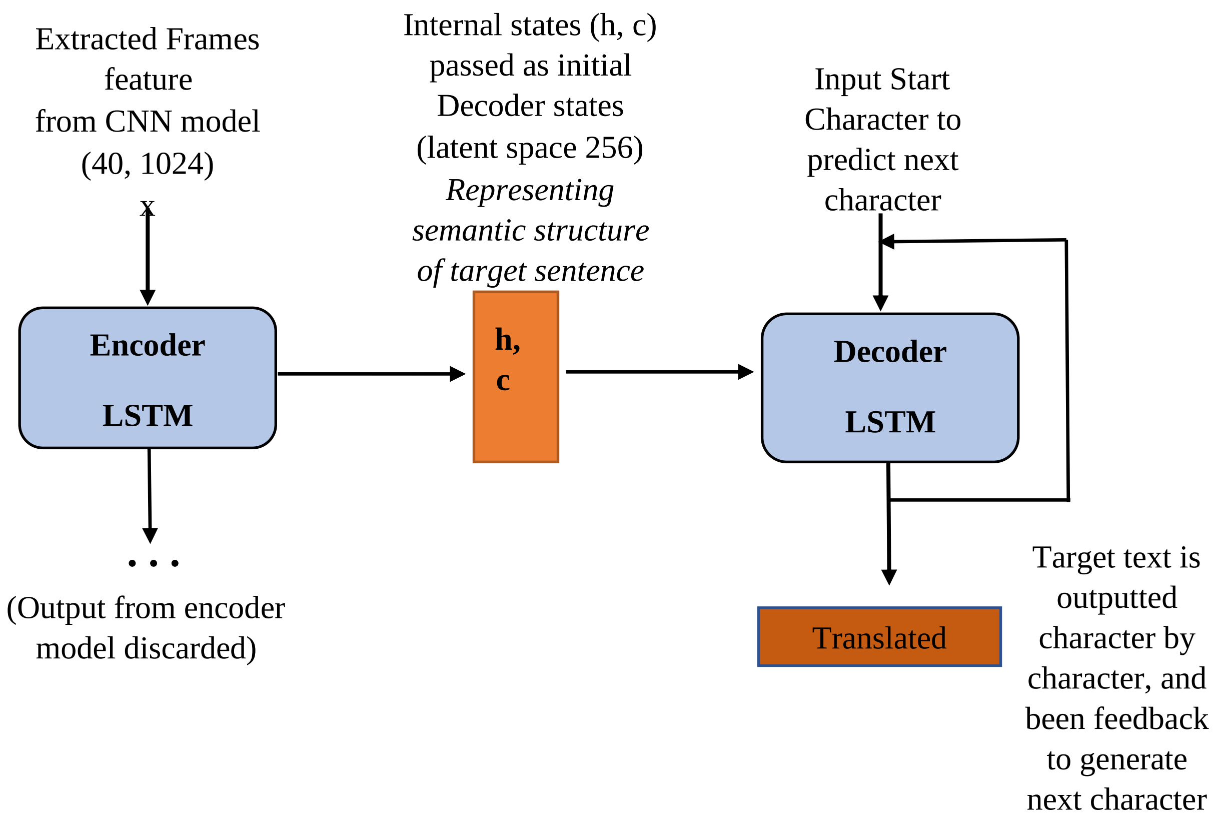

Extracted Frames feature from CNN model (40, 1024)
x
Internal states (h, c) passed as initial Decoder states (latent space 256)
Representing semantic structure of target sentence
Input Start Character to predict next character
Encoder LSTM
h, c
Decoder LSTM
. . .
(Output from encoder model discarded)
Translated
Target text is outputted character by character, and been feedback to generate next character

Figure 3.5: Rolled Schematic diagram of Encoder-Decoder Model

Chapter 4: Implementation

4.1 System Components

The Flow Chart Diagram in Figure 3.1 depicts the components of the entire system. These components include (1) Application Interface (2) Web Service (3) Data Processing Pipeline (4) Translation Engine.

4.2 Web Application Interface Implementation

The application interface is implemented on top of the Web Service as a Web Application, as well as a Mobile Application. These interfaces request data from the Web Service, format this data, and then displays the information for users.

Angular[1] Web Application Framework

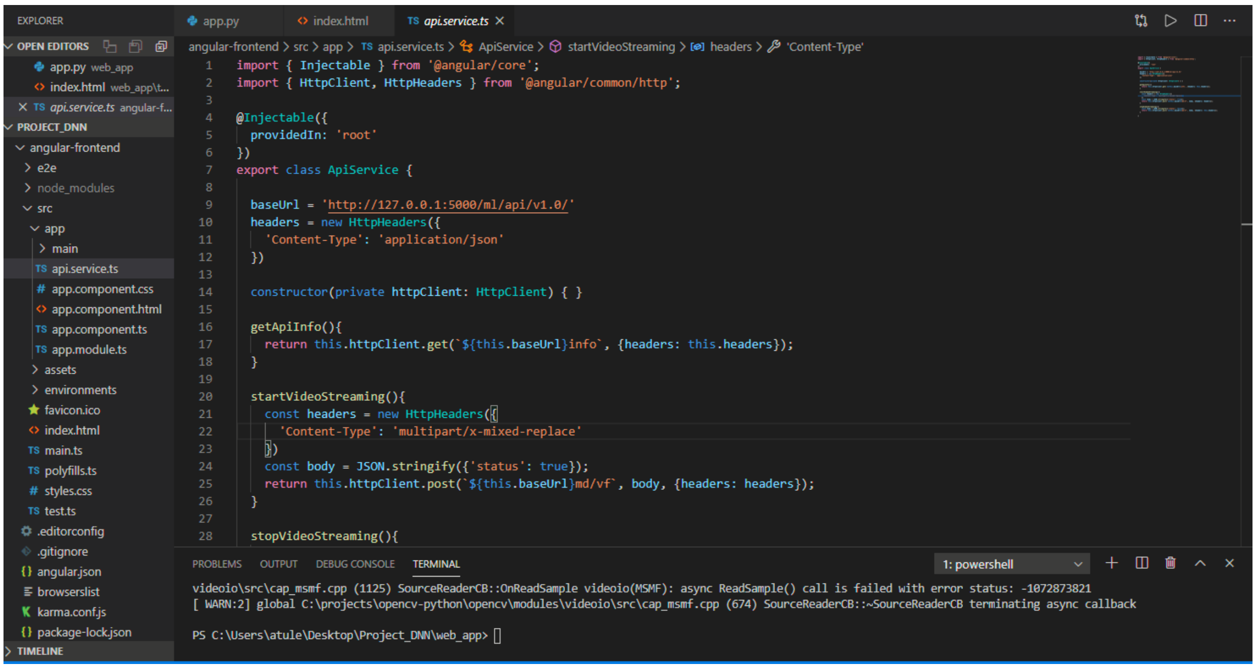
The Web Application Interface is implemented using the open-source Angular[1] web application framework maintained by Google, individuals, and other corporations. This framework is both mobile and desktop-ready, which means building interfaces for multiple platforms is easier. Furthermore, as a framework, as compared to using pure JavaScript, Angular comes with boilerplate code and components, which helps bootstrap the development process and reduces the development time.

Angular HttpClient[2] HTTP API

As seen from the schematic diagram of the Web Service, Figure 3.2**,** the web application interface is expected to communicate with the backend service by sending requests and receiving responses. The HttpClient[2] Module in the Angular framework offers a simplified HTTP API to enable communication with the backend Web Service. It contains

[1] Angular, https://angular.io/
[2] Angular HttpClient, https://angular.io/guide/http

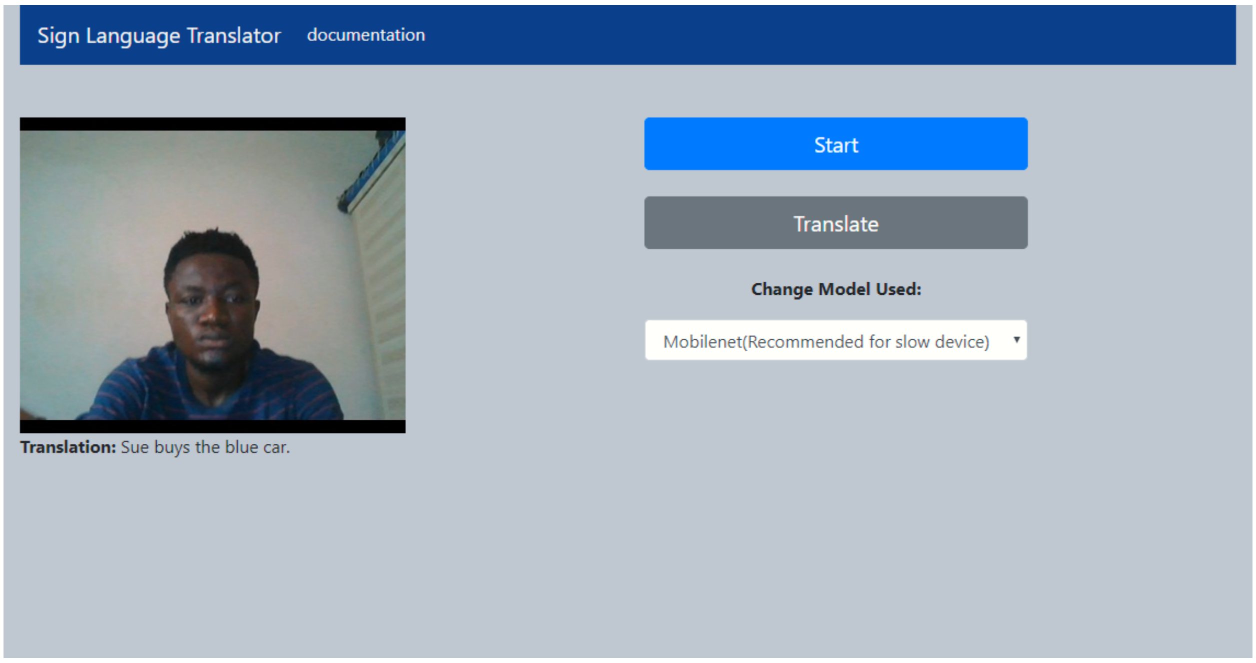
simple endpoints such as POST, GET, PUT etc. to create and send requests as well as receive generated responses from the backend service.


EXPLORER
app.py
index.html
api.service.ts
angular-frontend > src > app > api.service.ts > ApiService > startVideoStreaming > headers > 'Content-Type'
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
1: powershell

```
import { Injectable } from '@angular/core';
import { HttpClient, HttpHeaders } from '@angular/common/http';

@Injectable({
  providedIn: 'root'
})
export class ApiService {

  baseUrl = 'http://127.0.0.1:5000/ml/api/v1.0/'
  headers = new HttpHeaders({
    'Content-Type': 'application/json'
  })

  constructor(private httpClient: HttpClient) { }

  getApiInfo(){
    return this.httpClient.get(`${this.baseUrl}info`, {headers: this.headers});
  }

  startVideoStreaming(){
    const headers = new HttpHeaders({
      'Content-Type': 'multipart/x-mixed-replace'
    })
    const body = JSON.stringify({'status': true});
    return this.httpClient.post(`${this.baseUrl}md/vf`, body, {headers: headers});
  }

  stopVideoStreaming(){
```

```
videoio\src\cap_msmf.cpp (1125) SourceReaderCB::OnReadSample videoio(MSMF): async ReadSample() call is failed with error status: -1072873821
[ WARN:2] global C:\projects\opencv-python\opencv\modules\videoio\src\cap_msmf.cpp (674) SourceReaderCB::~SourceReaderCB terminating async callback

PS C:\Users\atule\Desktop\Project_DNN\web_app> 
```

Figure 4.1: Code showing Angular HttpClient accessing JSON data from web service



Translation: Sue buys the blue car.

Figure 4.2: Web application interface

4.3 Web Service (API) Implementation

The Web Service acts as the interface and facilitates the interaction between the application interfaces and the underlying Translation Engine of the system. This architecture was chosen to enable the decoupling of the application interfaces and the backend Translation Engine as required by the system design architecture. The Web Service is implemented in the Python[3] language, using the Flask[4] Micro Web Framework. OpenCV-Python[5], a python wrapper to OpenCV, is employed by the Web Service to access the camera feed of the users, in order to provide camera data of user sign language communication for prediction.

Python Language:

The Python programming language is an interpreted, high-level, general-purpose programming language. Python is best used for data-intensive processing and provides a vast number of free to use third-party modules as well as extensive support libraries to increase development and processing speed. Furthermore, the best Machine Learning Frameworks such as Keras[6], Tensorflow[7], and PyTorch are open-sourced and readily available in Python[3] to use. This, therefore, made the decision to use Python[3] inevitable.

Flask[4] Micro Web Framework:

Flask is a lightweight web application framework designed to make web development easy and quick to get started with, but with the capability to scale up to complex applications. It does not provide boilerplates, but contains minimal tools and libraries, in order to provide more flexibility to the application developer. The Flask

[3] Python, https://www.python.org/
[4] Flask, https://flask.palletsprojects.com/en/1.1.x/
[5] OpenCV-Python, https://pypi.org/project/opencv-python/
[6] Keras, https://keras.io/
[7] Tensorflow, https://www.tensorflow.org/

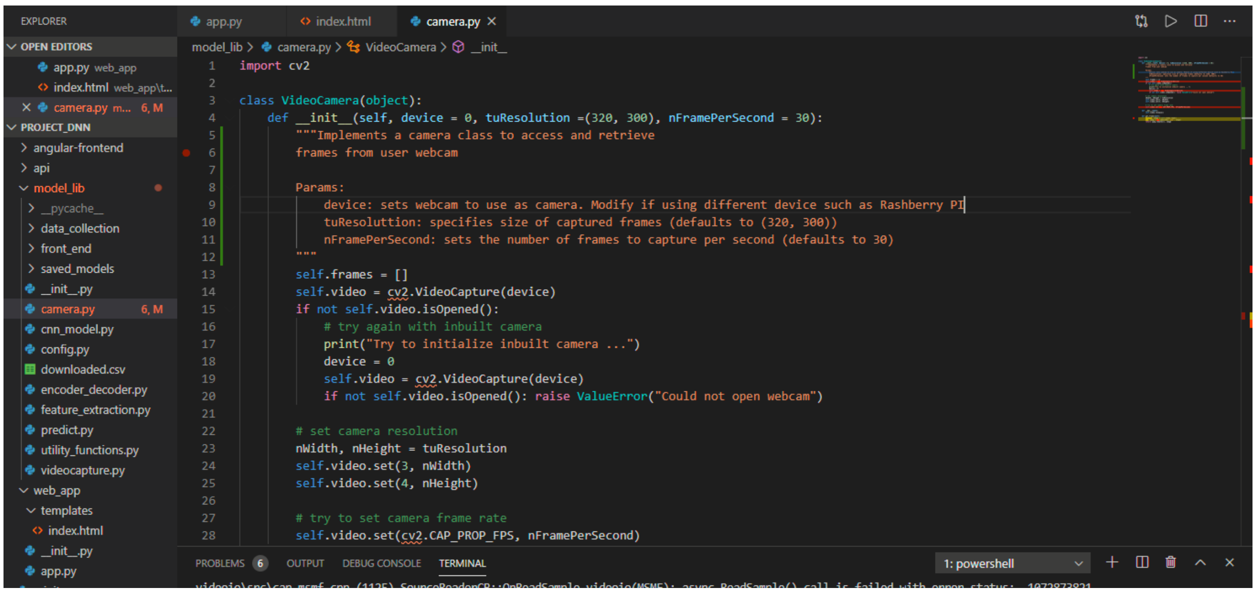
Framework is used in the development of the Web Service because of the ease of integration with the underlying Backend Translation Engine of the system since the Engine is also implemented using the Python Language. Furthermore, high flexibility provides total control over the development process.

OpenCV[9]

OpenCV is an image and video processing library with bindings in Python[3] and other programming languages. OpenCV is used for image and video analysis, as well for also capturing camera feed from a user device. OpenCV could also be used for facial and object detection, and other machine learning applications, however, the library is used for video capturing, and image analysis by the Web Service. OpenCV provides Python binding called OpenCV-Python in order to use it in a Python project.

The decision to use OpenCV coincides with the requirements of this project. It is not only freely available to use but is fast since it is implemented in C/C++ programming languages. Furthermore, OpenCV uses low RAM making it very efficient to handle and process the large data requirement of this application.

[9] OpenCV, https://opencv.org/



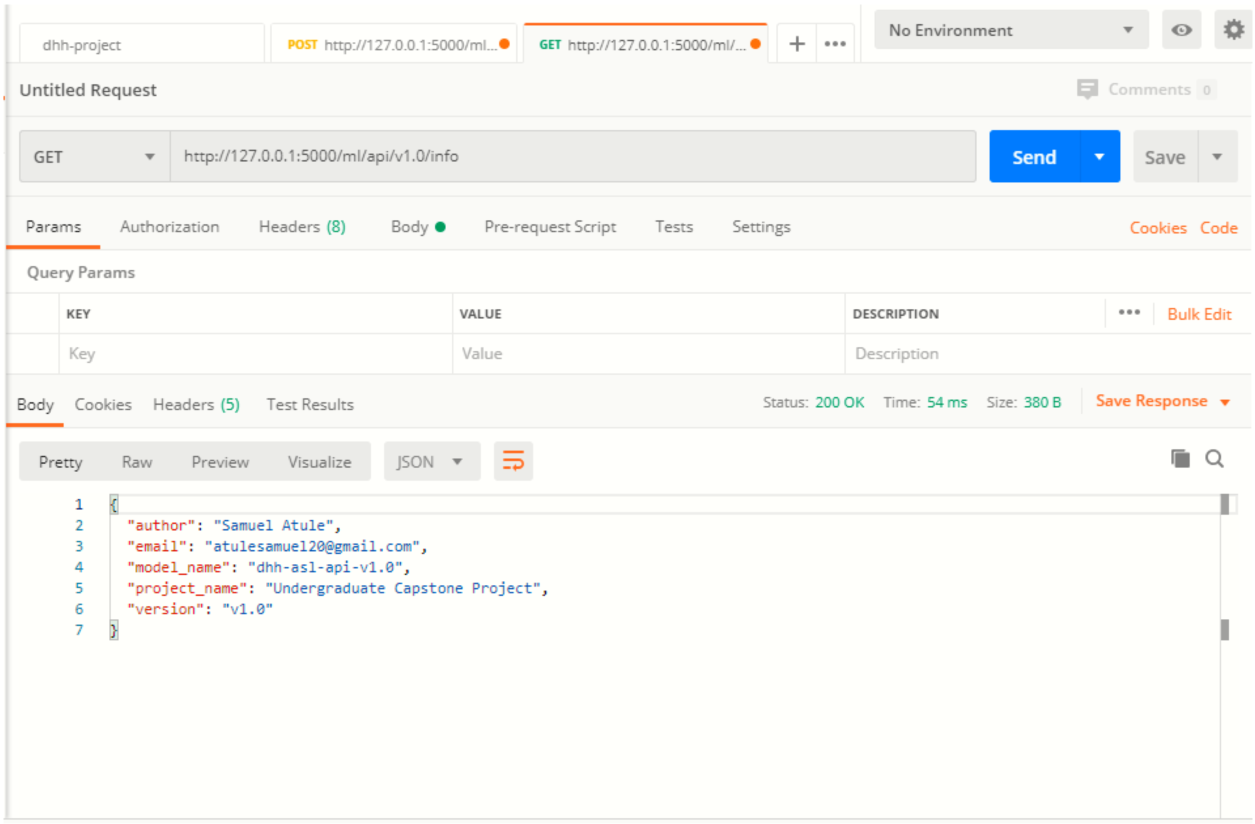
```python
import cv2

class VideoCamera(object):
    def __init__(self, device = 0, tuResolution =(320, 300), nFramePerSecond = 30):
        """Implements a camera class to access and retrieve
        frames from user webcam

        Params:
            device: sets webcam to use as camera. Modify if using different device such as Rashberry PI
            tuResoluttion: specifies size of captured frames (defaults to (320, 300))
            nFramePerSecond: sets the number of frames to capture per second (defaults to 30)
        """
        self.frames = []
        self.video = cv2.VideoCapture(device)
        if not self.video.isOpened():
            # try again with inbuilt camera
            print("Try to initialize inbuilt camera ...")
            device = 0
            self.video = cv2.VideoCapture(device)
            if not self.video.isOpened(): raise ValueError("Could not open webcam")

        # set camera resolution
        nWidth, nHeight = tuResolution
        self.video.set(3, nWidth)
        self.video.set(4, nHeight)

        # try to set camera frame rate
        self.video.set(cv2.CAP_PROP_FPS, nFramePerSecond)
```

Figure 4.3: Code showing implementation of OpenCV to access frames from the webcam

Postman[10]

Postman is an HTTP API testing tool, which was used to test the various endpoints of the web service. It provides an intuitive graphical user interface (GUI), and standard HTTP methods to make testing APIs easier.

[10] https://www.postman.com/

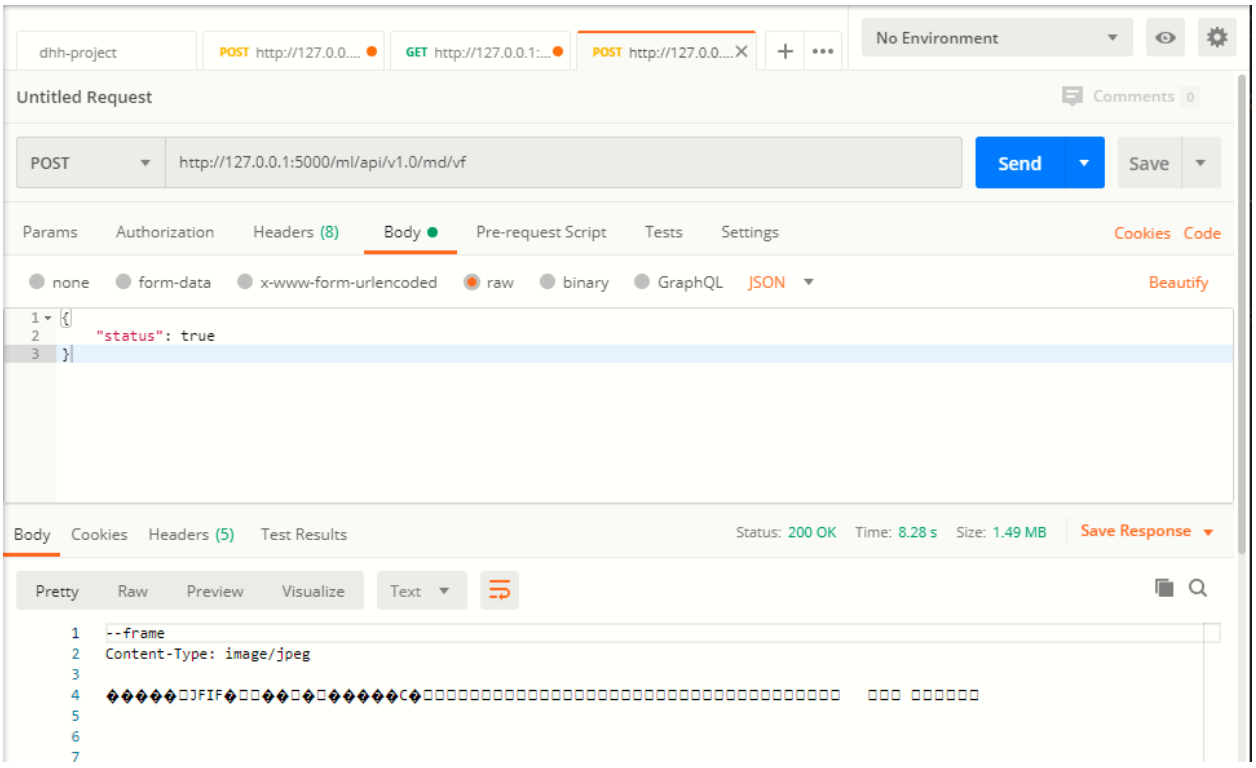


GET http://127.0.0.1:5000/ml/api/v1.0/info

Status: 200 OK Time: 54 ms Size: 380 B

```
{
  "author": "Samuel Atule",
  "email": "atulesamuel20@gmail.com",
  "model_name": "dhh-asl-api-v1.0",
  "project_name": "Undergraduate Capstone Project",
  "version": "v1.0"
}
```

Figure 4.4: Postman sending a get request for API information



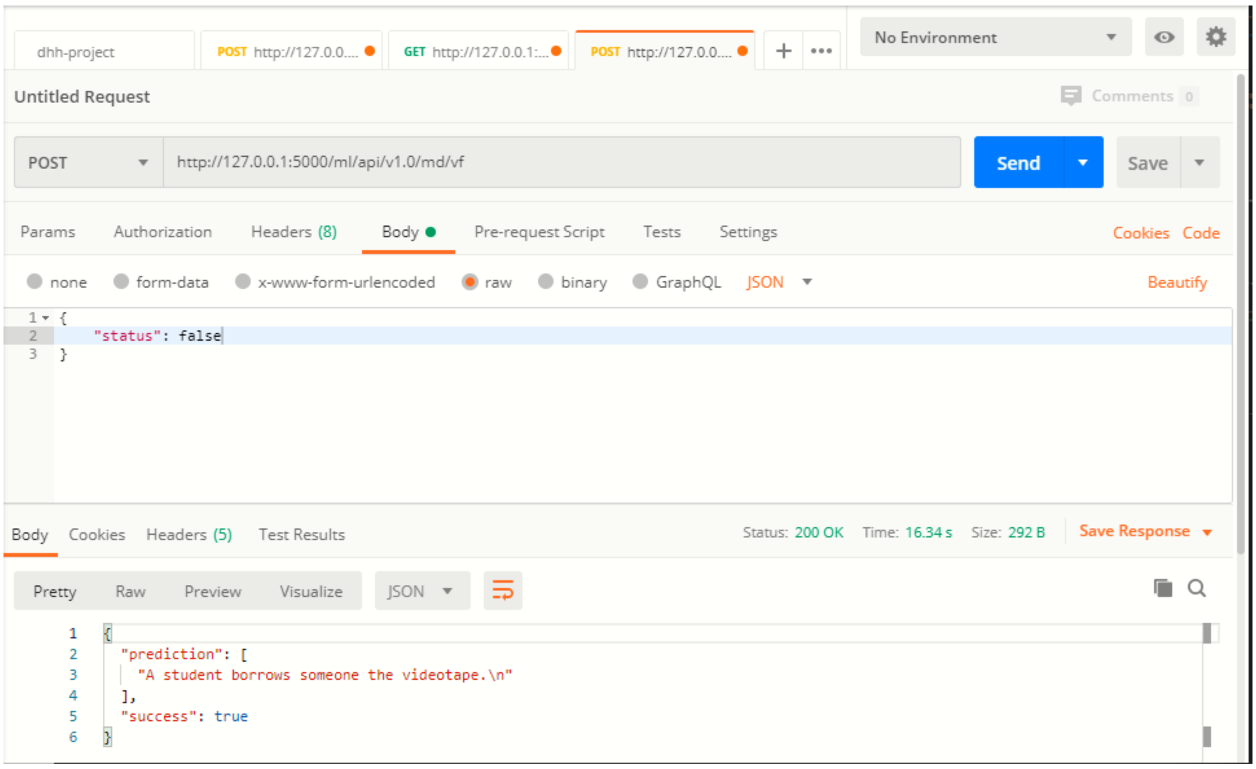
POST http://127.0.0.1:5000/ml/api/v1.0/md/vf

```
{
    "status": true
}
```

Status: 200 OK Time: 8.28 s Size: 1.49 MB

```
--frame
Content-Type: image/jpeg

�����JFIF����������C�
```

Figure 4.5: Postman sending a post request to stream frames from the webcam



POST http://127.0.0.1:5000/ml/api/v1.0/md/vf Send

```
{
    "status": false
}
```

Status: 200 OK Time: 16.34 s Size: 292 B

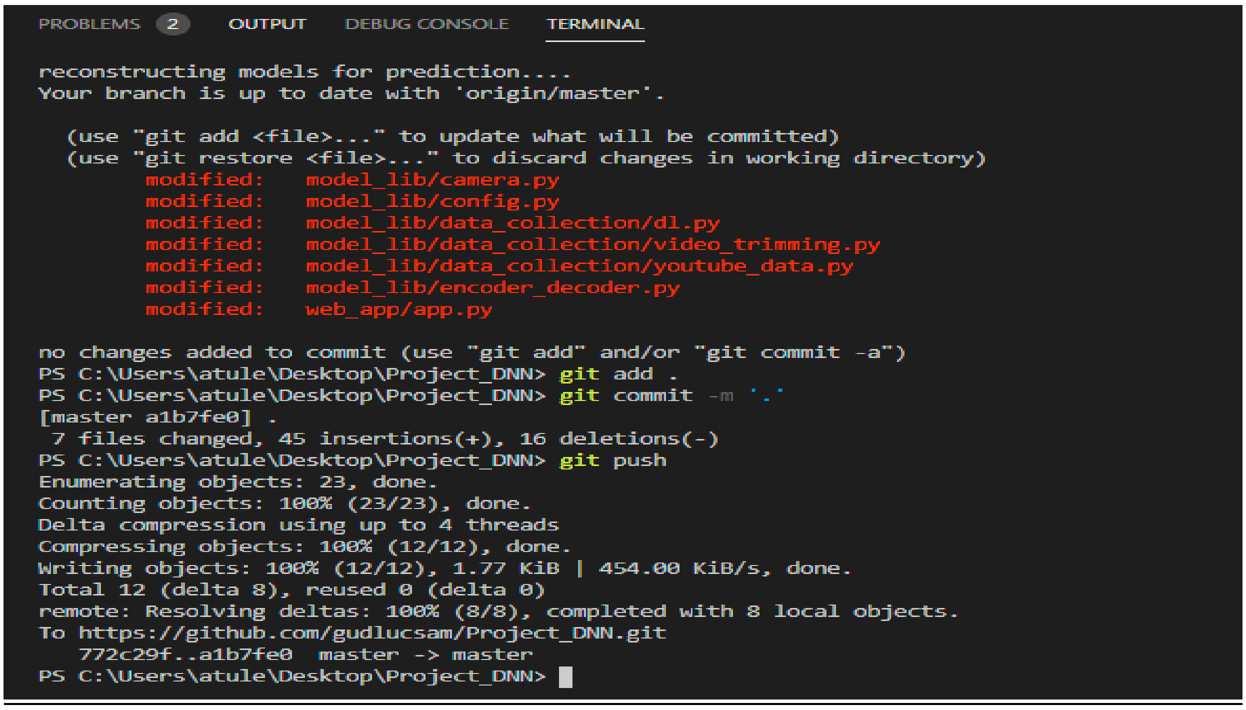
```
{
  "prediction": [
    "A student borrows someone the videotape.\n"
  ],
  "success": true
}
```

Figure 4.6: Sending a post request with frames for prediction in Postman

Git[11] and GitHub[12]

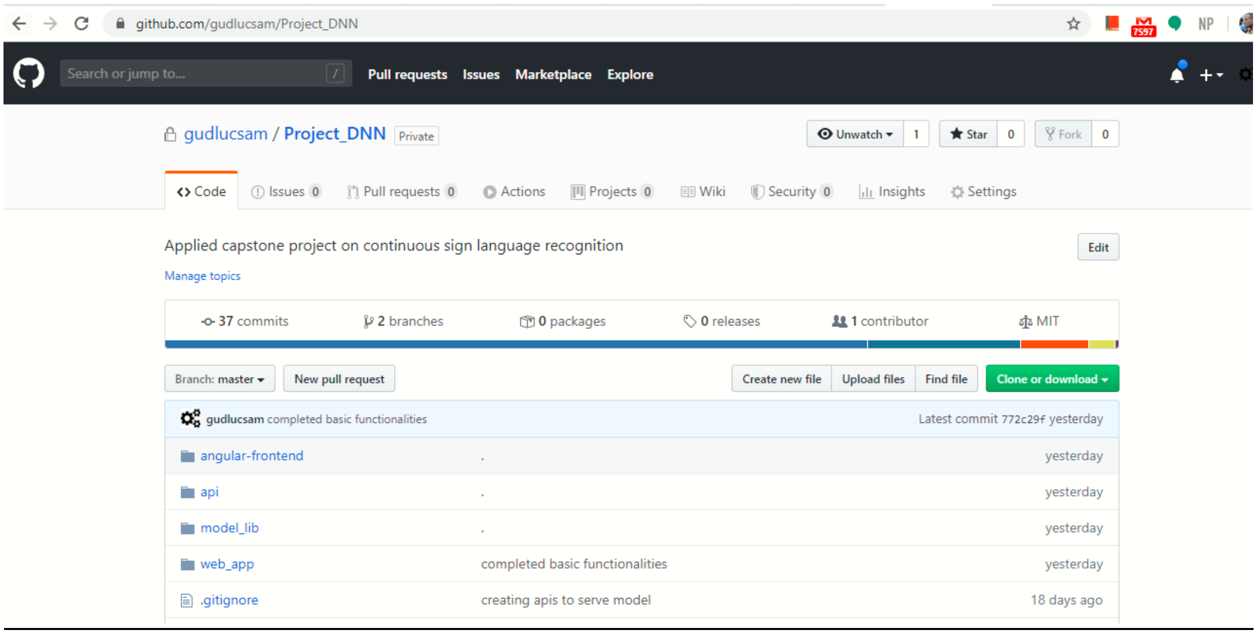
Git is an open-source version-control system that was used during the development process to track changes in the project source code. This helped to maintain project code by creating different git branches to implement separate logic before merging to the central repository. Furthermore, the source code was published unto a remote GitHub[12] repository to avoid losing code if the local computer is not available. GitHub[12] provides remote hosting for version control systems.

[11] Git, https://git-scm.com/
[12] Github, https://github.com/



```
reconstructing models for prediction....
Your branch is up to date with 'origin/master'.

  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   model_lib/camera.py
        modified:   model_lib/config.py
        modified:   model_lib/data_collection/dl.py
        modified:   model_lib/data_collection/video_trimming.py
        modified:   model_lib/data_collection/youtube_data.py
        modified:   model_lib/encoder_decoder.py
        modified:   web_app/app.py

no changes added to commit (use "git add" and/or "git commit -a")
PS C:\Users\atule\Desktop\Project_DNN> git add .
PS C:\Users\atule\Desktop\Project_DNN> git commit -m '.'
[master a1b7fe0] .
 7 files changed, 45 insertions(+), 16 deletions(-)
PS C:\Users\atule\Desktop\Project_DNN> git push
Enumerating objects: 23, done.
Counting objects: 100% (23/23), done.
Delta compression using up to 4 threads
Compressing objects: 100% (12/12), done.
Writing objects: 100% (12/12), 1.77 KiB | 454.00 KiB/s, done.
Total 12 (delta 8), reused 0 (delta 0)
remote: Resolving deltas: 100% (8/8), completed with 8 local objects.
To https://github.com/gudlucsam/Project_DNN.git
   772c29f..a1b7fe0  master -> master
PS C:\Users\atule\Desktop\Project_DNN> 
```

Figure 4.7: Output of pushing changes to Github using Git



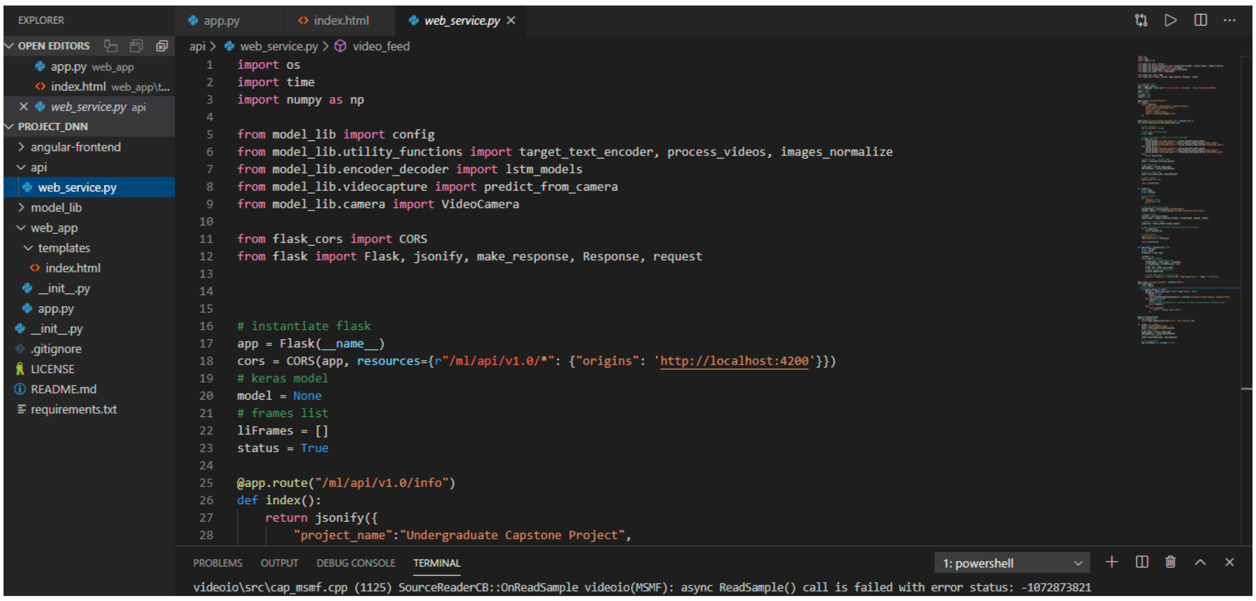
Applied capstone project on continuous sign language recognition

37 commits   2 branches   0 packages   0 releases   1 contributor   MIT

gudlucsam completed basic functionalities   Latest commit 772c29f yesterday

angular-frontend	.	yesterday
api	.	yesterday
model_lib	.	yesterday
web_app	completed basic functionalities	yesterday
.gitignore	creating apis to serve model	18 days ago

Figure 4.8: Source code published on GitHub



```python
import os
import time
import numpy as np

from model_lib import config
from model_lib.utility_functions import target_text_encoder, process_videos, images_normalize
from model_lib.encoder_decoder import lstm_models
from model_lib.videocapture import predict_from_camera
from model_lib.camera import VideoCamera

from flask_cors import CORS
from flask import Flask, jsonify, make_response, Response, request



# instantiate flask
app = Flask(__name__)
cors = CORS(app, resources={r"/ml/api/v1.0/*": {"origins": 'http://localhost:4200'}})
# keras model
model = None
# frames list
liFrames = []
status = True

@app.route("/ml/api/v1.0/info")
def index():
    return jsonify({
        "project_name":"Undergraduate Capstone Project",
```

```
videoio\src\cap_msmf.cpp (1125) SourceReaderCB::OnReadSample videoio(MSMF): async ReadSample() call is failed with error status: -1072873821
```

Figure 4.9: Code showing implementation of Web Service in Visual Studio Code

4.4 Data Collection and Processing Pipeline

The Data Collection and Data processing Pipeline is a crucial component needed to implement the Translation Model. The Translation Model requires a cornucopia of video data in order to be able to provide correct results. Furthermore, this module also transforms the video data into a ready-consumable format for the Translation Model.

YouTube API[13]

The YouTube API allows developers and programmers to access video statistics and YouTube channels data via REST and XML-RPC calls. This API[13] was employed to harvest and create a large volume of training data for the Translation Model since there is a dearth of training data for Sign Language Translation. Playlist videos from the American Sign Language Youtube Channel called ASLMeredith[14] was downloaded together with their corresponding transcription. The videos are then segmented using the sentence durations of its transcription in JSON format. Subsequently, manual cross-checking was carried out to

[13] YouTube API, https://pypi.org/project/python-youtube/
[14] ASLMeredith, https://www.youtube.com/channel/UCsuBAbMlPGJ95w3mwhz-30g

remove bad data points before the generated dataset is fed into the Data Processing Pipeline. This API is a good choice because it is implemented in the Python programming language and integrates seamlessly with the entire system code for automation. Furthermore, Youtube videos were free to download without any legal restriction leading to a short amount of time in generation a large dataset.

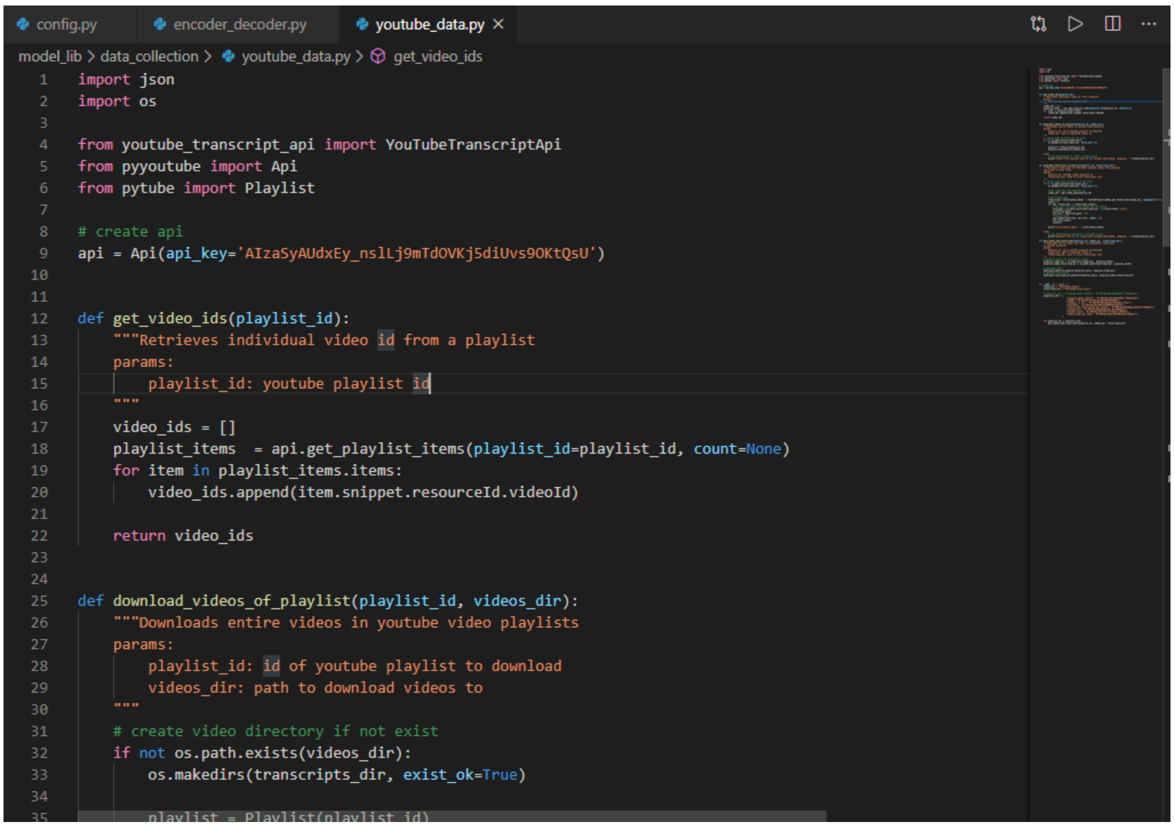
MoviePy[15] Python Module

MoviePy is a Python module used for video editing, such as cutting, concatenating, or title insertions. It is also used for video compositing, video processing, and works with the most common video formats such as MPEG, mp4, GIF, among others. In order to generate the training data from the YouTube playlists, this module was employed to trim video segments into sentence long duration using their corresponding transcript data. This module was used because it supports the most video formats such as MPEG and mp4, which the downloaded YouTube videos used.

Pytube3[16] Python Module

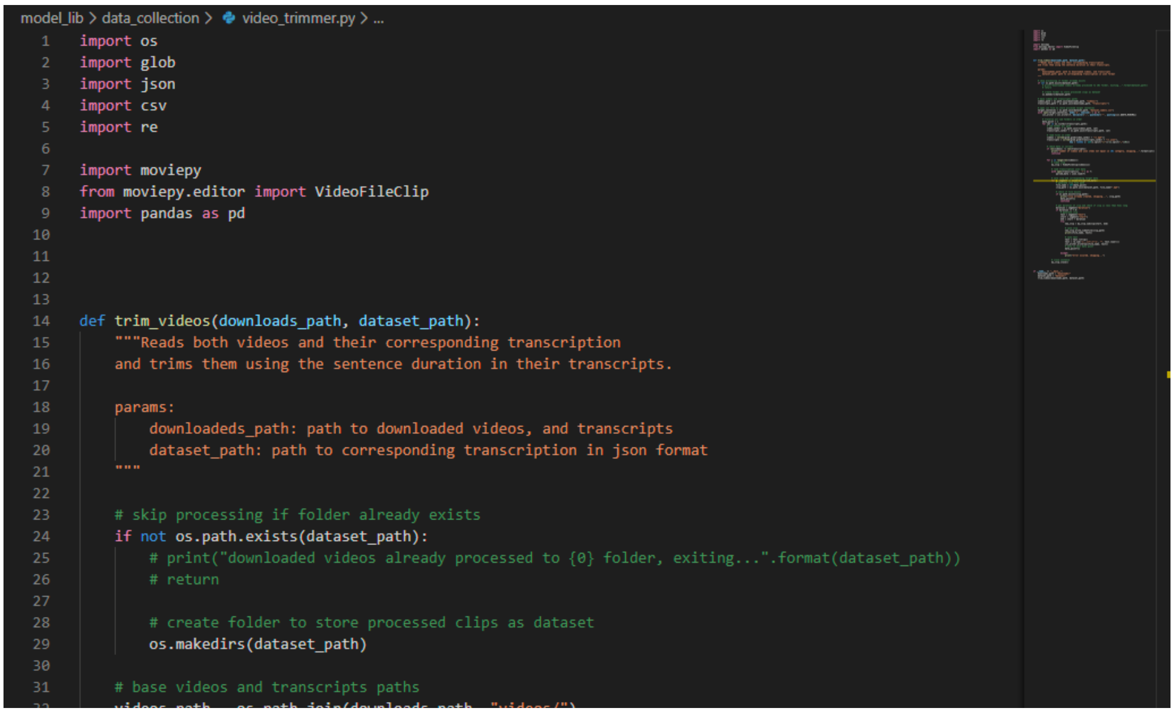
Pytube3 is a lightweight Python library for downloading YouTube videos. It is easy to configure and use and provide interfaces for downloading video playlists on YouTube.

[15] MoviePy, https://pypi.org/project/moviepy/
[16] Pytube3, https://python-pytube.readthedocs.io/en/latest/



```python
import json
import os

from youtube_transcript_api import YouTubeTranscriptApi
from pyyoutube import Api
from pytube import Playlist

# create api
api = Api(api_key='AIzaSyAUdxEy_nslLj9mTdOVKj5diUvs9OKtQsU')


def get_video_ids(playlist_id):
    """Retrieves individual video id from a playlist
    params:
        playlist_id: youtube playlist id
    """
    video_ids = []
    playlist_items  = api.get_playlist_items(playlist_id=playlist_id, count=None)
    for item in playlist_items.items:
        video_ids.append(item.snippet.resourceId.videoId)

    return video_ids


def download_videos_of_playlist(playlist_id, videos_dir):
    """Downloads entire videos in youtube video playlists
    params:
        playlist_id: id of youtube playlist to download
        videos_dir: path to download videos to
    """
    # create video directory if not exist
    if not os.path.exists(videos_dir):
        os.makedirs(transcripts_dir, exist_ok=True)

        playlist = Playlist(playlist_id)
```

Figure 4.10: Using YouTube API to download sign language playlists and transcriptions

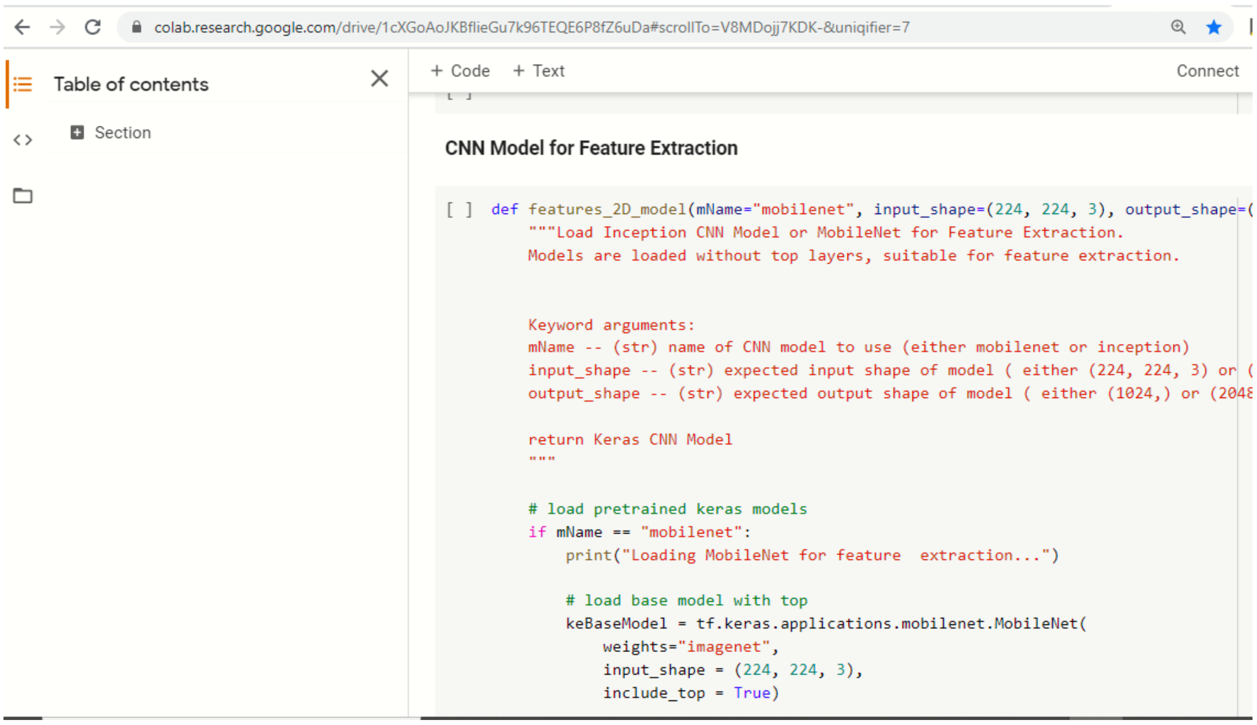


```python
import os
import glob
import json
import csv
import re

import moviepy
from moviepy.editor import VideoFileClip
import pandas as pd




def trim_videos(downloads_path, dataset_path):
    """Reads both videos and their corresponding transcription
    and trims them using the sentence duration in their transcripts.

    params:
        downloadeds_path: path to downloaded videos, and transcripts
        dataset_path: path to corresponding transcription in json format
    """

    # skip processing if folder already exists
    if not os.path.exists(dataset_path):
        # print("downloaded videos already processed to {0} folder, exiting...".format(dataset_path))
        # return

        # create folder to store processed clips as dataset
        os.makedirs(dataset_path)

    # base videos and transcripts paths
```

Figure 4.11: Using MoviePy to trim videos to sentence-level durations

Google Collaboratory[17]

Google Collaboratory is a free cloud service based on the Jupiter Notebook and supports free GPU and 12GB RAM. Is also comes with most machine learning modules already installed. This platform was used to train the machine learning model for translation.



colab.research.google.com/drive/1cXGoAoJKBflieGu7k96TEQE6P8fZ6uDa#scrollTo=V8MDojj7KDK-&uniqifier=7
Table of contents
Section
+ Code
+ Text
Connect
CNN Model for Feature Extraction

```
[ ] def features_2D_model(mName="mobilenet", input_shape=(224, 224, 3), output_shape=(
        """Load Inception CNN Model or MobileNet for Feature Extraction.
        Models are loaded without top layers, suitable for feature extraction.


        Keyword arguments:
        mName -- (str) name of CNN model to use (either mobilenet or inception)
        input_shape -- (str) expected input shape of model ( either (224, 224, 3) or (
        output_shape -- (str) expected output shape of model ( either (1024,) or (2048

        return Keras CNN Model
        """

        # load pretrained keras models
        if mName == "mobilenet":
            print("Loading MobileNet for feature  extraction...")

            # load base model with top
            keBaseModel = tf.keras.applications.mobilenet.MobileNet(
                weights="imagenet",
                input_shape = (224, 224, 3),
                include_top = True)
```

Figure 4.12: Source code on Google Collaboratory

Pandas[18]

Pandas is an open-source library that provides high-performance, easy-to-use data structures and data analysis tools for the Python programming language. This library was used to format text data in the data processing pipeline.

NumPy[19]

[17] Google Collaboratory, https://research.google.com/colaboratory/faq.html
[18] Pandas, https://pandas.pydata.org/
[19] NumPy, https://numpy.org/

NumPy is an open-source library for scientific computing with Python. This library was employed because of the high tensor operation requirements of video data. Furthermore, it is also employed by all the Machine Learning libraries used in the Translation Engine Model.

4.5 Sign Language Translation Model Implementation

This component is the backbone of the entire system. It accepts user's sign language communication in the form of video frames and computes the translation of the sign language into the English Language, using Deep Learning. This component employs several machine learning technologies and state of the art machine learning techniques such as Keras, TensorFlow, and Transfer Learning.

General Implementation Technique

As indicated in the model architecture, after the video frames are extracted and processed, they are then passed into one of two Convolutional Neural Network: the MobileNet or InceptionV3 for feature extraction. The choice of including these models was to provide preference depending on the nature of the user's device. For instance, MobileNet is a lightweight model, and will preferably run on a low-end user device. These extracted features are then fed into an Encoder Model, a Long Short-Term Memory (LSTM) Recurrent Neural Network (RNN), to compute a 256-sized latent space vector representing the semantic space of the expected output sentence. The 256-sized latent space vector is then used as the initial states for another LSTM-RNN Model called the Decoder Model. This model then outputs sentence character by character using the tab character as a starting character and then feeding the previously generated character back to the Decoder Model to output the next character. Refer to the general architecture and the motivation for using Transfer Learning.

InceptionV3[20]

InceptionV3[20] is a convolutional neural network model made available in Keras alongside its weights and can be used for prediction, feature extraction, and fine-tuning. This model is heavy with a size of 92MB, with over 23 million configurable parameters. However, it has a top-5 accuracy of 0.937 on the ImageNet validation dataset.

MobileNet[21]

MobileNet is also another deep learning model made available in Keras alongside its weights and can be used for prediction, feature extraction, and fine-tuning. Unlike InceptionV3, this model is lightweight with a size of 16MB, and about 4 million configurable parameters. However, it has a top-5 accuracy of 0.895 on the ImageNet validation dataset.

Keras[7]

Keras is an open-sourced machine learning framework written in the Python programming language. Keras is easy to use, largely supported and makes configuring neural networks less complicated. It is a high-level API implemented on top of several high-end numeral computational libraries such as TensorFlow Core API, Theano, CNTK, among others. This framework was chosen because of its ease of building models, as well as its support of most of the backend numeric computational engines.

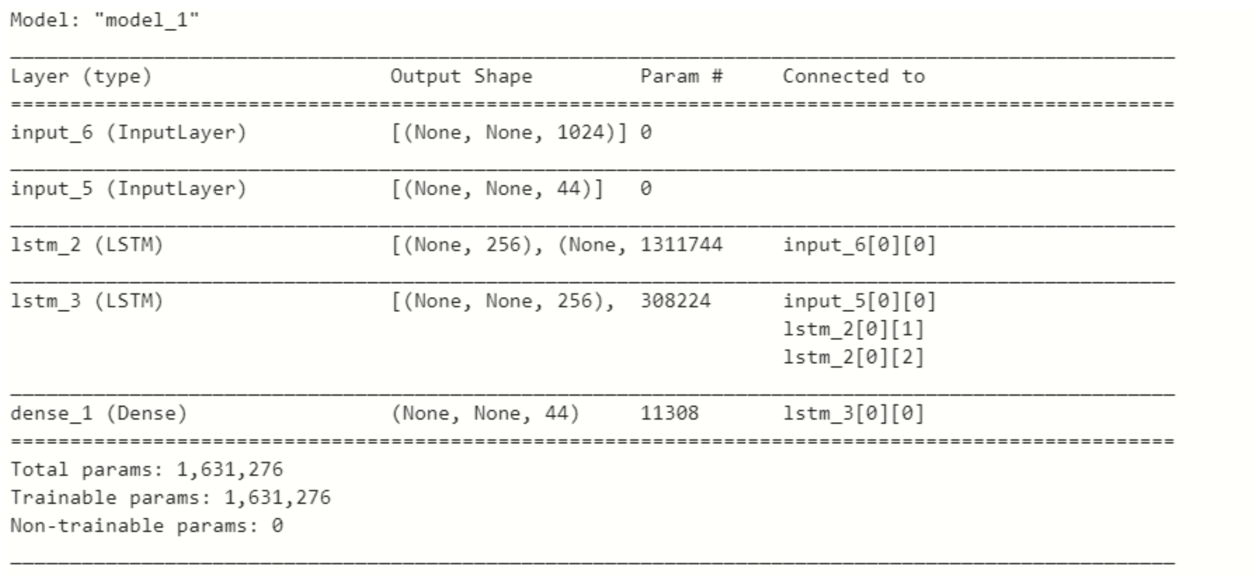
TensorFlow[9]

The TensorFlow Library is an open-source for symbolic math operations and used for machine learning applications such as building neural networks. This library was used with the Keras framework for fast vector operations involving the video data.

Encoder-Decoder Models

[20] InceptionV3, https://keras.io/api/applications/inceptionv3/
[21] MobileNet, https://keras.io/api/applications/mobilenet/

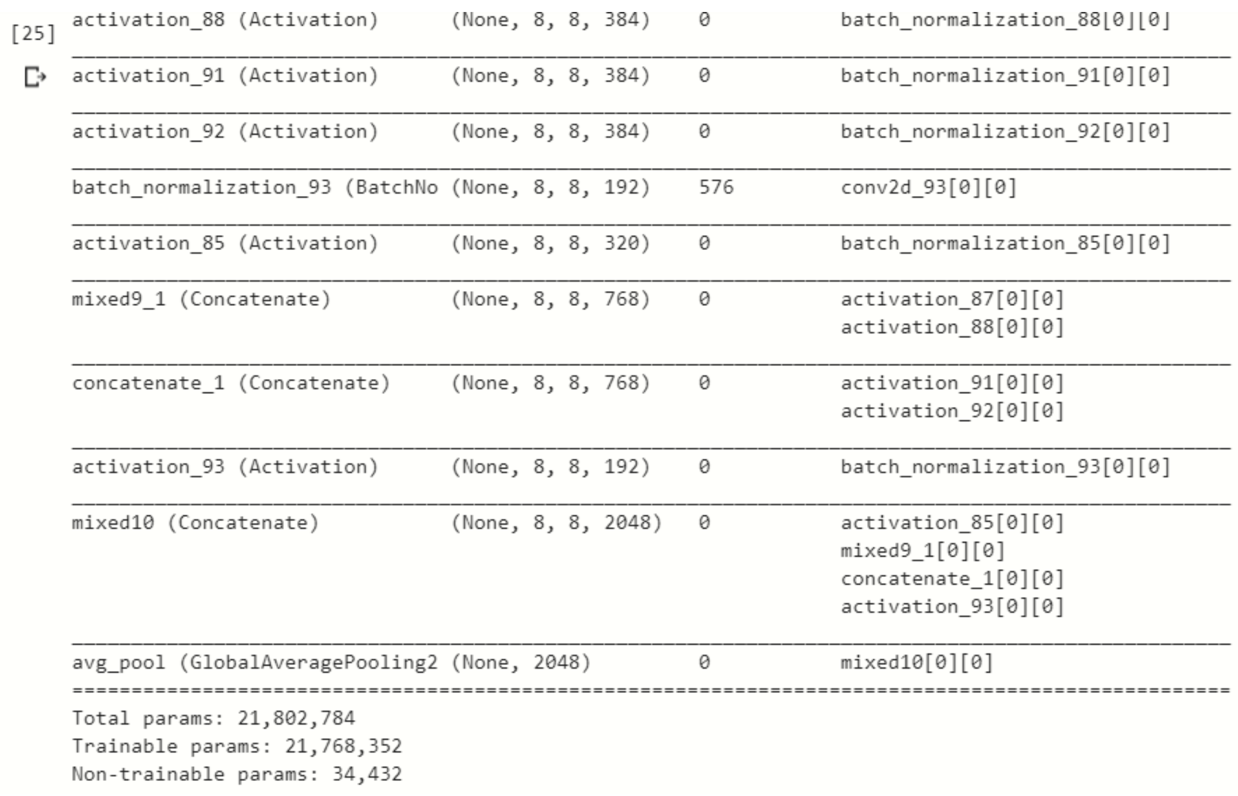
The Encoder-Decoderer Model implemented in this system is an adaption of the Sequence to Sequence Model example implemented on the official Keras documentation page[22].



```
Model: "model_1"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_6 (InputLayer)            [(None, None, 1024)] 0
__________________________________________________________________________________________________
input_5 (InputLayer)            [(None, None, 44)]   0
__________________________________________________________________________________________________
lstm_2 (LSTM)                   [(None, 256), (None, 1311744     input_6[0][0]
__________________________________________________________________________________________________
lstm_3 (LSTM)                   [(None, None, 256),  308224      input_5[0][0]
                                                                 lstm_2[0][1]
                                                                 lstm_2[0][2]
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, None, 44)     11308       lstm_3[0][0]
==================================================================================================
Total params: 1,631,276
Trainable params: 1,631,276
Non-trainable params: 0
__________________________________________________________________________________________________
```

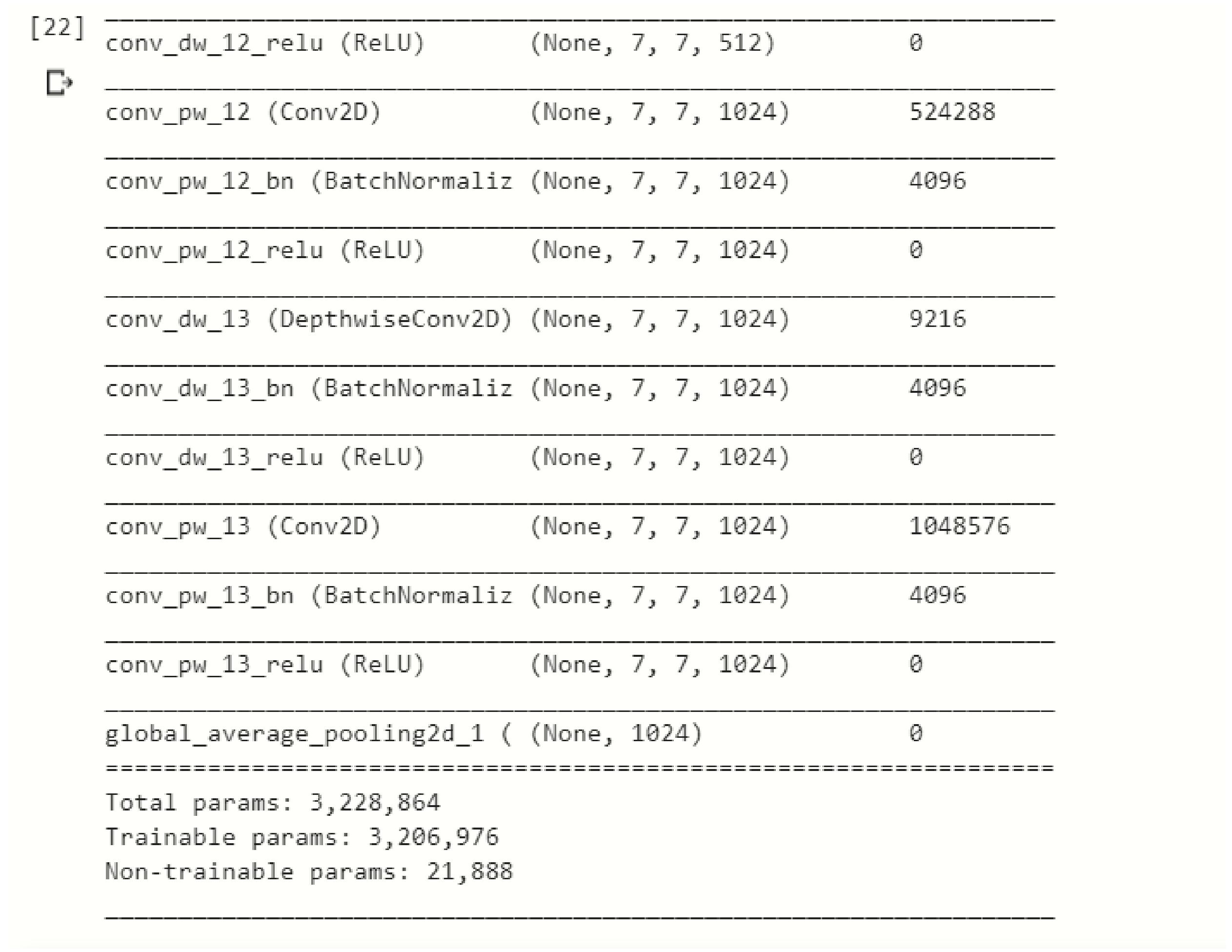
Figure 4.13: Visualization of Encoder-Decoder architecture

[22] Sequence to Sequence Model Example, https://keras.io/examples/lstm_seq2seq/



```
[25] activation_88 (Activation)      (None, 8, 8, 384)    0          batch_normalization_88[0][0]
     ________________________________________________________________________________________________
     activation_91 (Activation)      (None, 8, 8, 384)    0          batch_normalization_91[0][0]
     ________________________________________________________________________________________________
     activation_92 (Activation)      (None, 8, 8, 384)    0          batch_normalization_92[0][0]
     ________________________________________________________________________________________________
     batch_normalization_93 (BatchNo (None, 8, 8, 192)    576        conv2d_93[0][0]
     ________________________________________________________________________________________________
     activation_85 (Activation)      (None, 8, 8, 320)    0          batch_normalization_85[0][0]
     ________________________________________________________________________________________________
     mixed9_1 (Concatenate)          (None, 8, 8, 768)    0          activation_87[0][0]
                                                                     activation_88[0][0]
     ________________________________________________________________________________________________
     concatenate_1 (Concatenate)     (None, 8, 8, 768)    0          activation_91[0][0]
                                                                     activation_92[0][0]
     ________________________________________________________________________________________________
     activation_93 (Activation)      (None, 8, 8, 192)    0          batch_normalization_93[0][0]
     ________________________________________________________________________________________________
     mixed10 (Concatenate)           (None, 8, 8, 2048)   0          activation_85[0][0]
                                                                     mixed9_1[0][0]
                                                                     concatenate_1[0][0]
                                                                     activation_93[0][0]
     ________________________________________________________________________________________________
     avg_pool (GlobalAveragePooling2 (None, 2048)         0          mixed10[0][0]
     ================================================================================================
     Total params: 21,802,784
     Trainable params: 21,768,352
     Non-trainable params: 34,432
```

Figure 4.14: Visualization of MobileNet CNN architecture



```
[22] _________________________________________________________________
     conv_dw_12_relu (ReLU)       (None, 7, 7, 512)         0
     _________________________________________________________________
     conv_pw_12 (Conv2D)          (None, 7, 7, 1024)        524288
     _________________________________________________________________
     conv_pw_12_bn (BatchNormaliz (None, 7, 7, 1024)        4096
     _________________________________________________________________
     conv_pw_12_relu (ReLU)       (None, 7, 7, 1024)        0
     _________________________________________________________________
     conv_dw_13 (DepthwiseConv2D) (None, 7, 7, 1024)        9216
     _________________________________________________________________
     conv_dw_13_bn (BatchNormaliz (None, 7, 7, 1024)        4096
     _________________________________________________________________
     conv_dw_13_relu (ReLU)       (None, 7, 7, 1024)        0
     _________________________________________________________________
     conv_pw_13 (Conv2D)          (None, 7, 7, 1024)        1048576
     _________________________________________________________________
     conv_pw_13_bn (BatchNormaliz (None, 7, 7, 1024)        4096
     _________________________________________________________________
     conv_pw_13_relu (ReLU)       (None, 7, 7, 1024)        0
     _________________________________________________________________
     global_average_pooling2d_1 ( (None, 1024)              0
     =================================================================
     Total params: 3,228,864
     Trainable params: 3,206,976
     Non-trainable params: 21,888
     _________________________________________________________________
```

Figure 4.15: Visualization of InceptionV3 CNN architecture

Chapter 5: Testing and Results

This chapter explores suitable testing techniques to validate the entire system against the functional and non-functional requirements of the project. Testing also reveals bugs and logical errors within the system. In order to assess the validity of the entire system, the following testing techniques where employed: Component Testing, System-level Testing, User Testing. In the final part of this chapter, an analysis of the test results, as well as the various part of the system that did not work out, is also stated for future implementation.

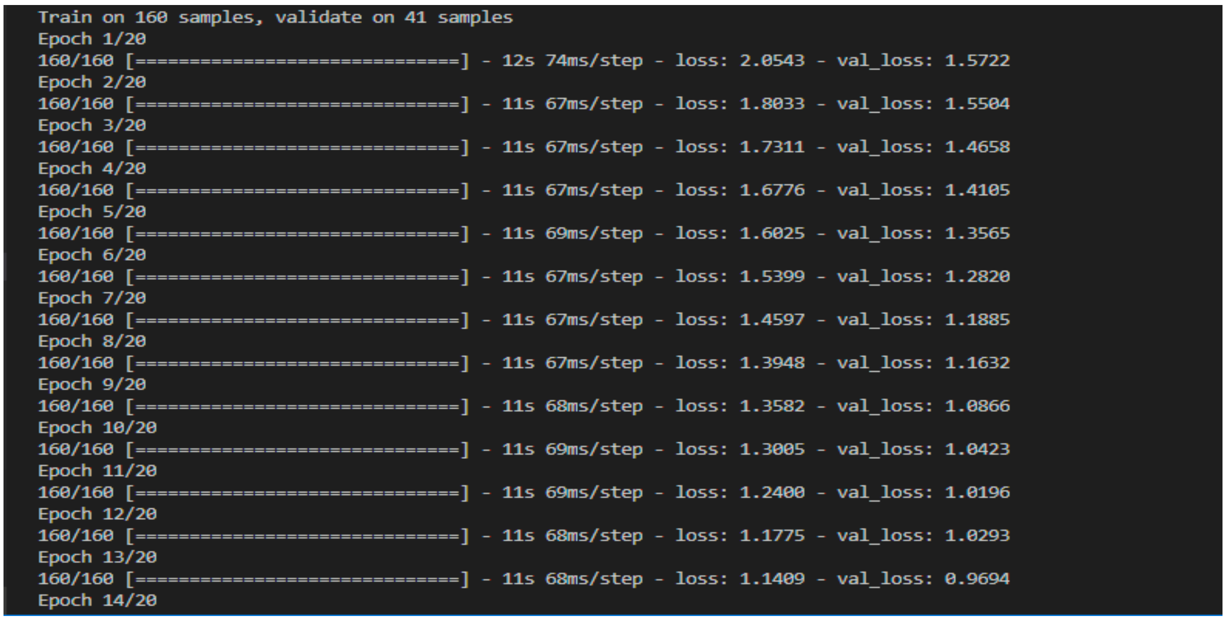
5.1 Development and Component Testing

Development Testing is a testing practice where testing is carried out alongside the development process. In this technique, the testing and the development process is tightly coupled together, such that the code is immediately checked as it is written. This means bugs and logic errors are quickly exposed, and the quality of the development code is improved. All the various system components where testing during the development phase to ensure that completed components were free of bugs.

Component testing was simultaneously carried during the development phase. In this testing technique, individual components of the system called modules are tested separately without integration with other components of the system. Since the business logic of the entire system was separated, it provided room to do Component Testing.

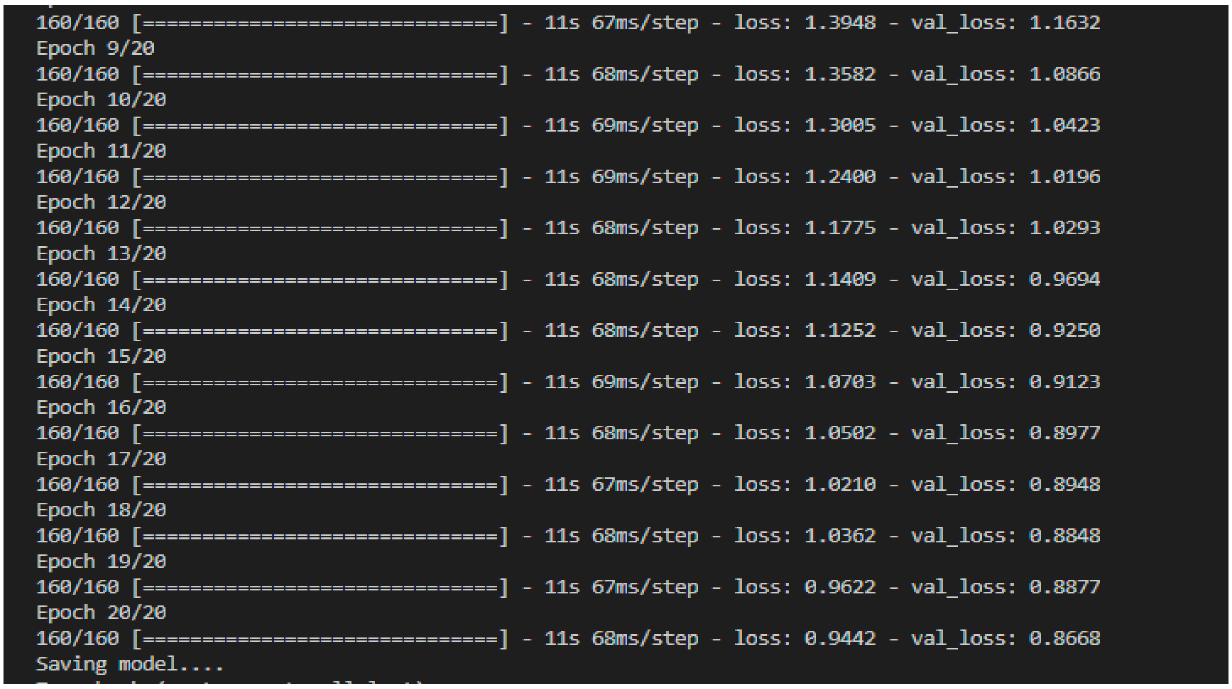
(i) Translation Backend Model Component

The Translation Model is a machine learning model and hence was tested to ensure the predictive accuracy of the model. During the development process, the model was trained on 161 video data points and validated on 40 video data points with 20 epochs. The Word Error Rate (WER) or the Character Error Rate (CER) are the two most suitable performance metrics to validate a translation model. However, due to the challenge of integrating a custom CER custom function with Keras framework, the validation accuracy, as well as empirical evidence of the model in use, was used as a performance metric. The CER custom metric was be implemented in future works of this project. The model was trained using a small epochs value of 20 because of the limited computing power available.



```
Train on 160 samples, validate on 41 samples
Epoch 1/20
160/160 [==============================] - 12s 74ms/step - loss: 2.0543 - val_loss: 1.5722
Epoch 2/20
160/160 [==============================] - 11s 67ms/step - loss: 1.8033 - val_loss: 1.5504
Epoch 3/20
160/160 [==============================] - 11s 67ms/step - loss: 1.7311 - val_loss: 1.4658
Epoch 4/20
160/160 [==============================] - 11s 67ms/step - loss: 1.6776 - val_loss: 1.4105
Epoch 5/20
160/160 [==============================] - 11s 69ms/step - loss: 1.6025 - val_loss: 1.3565
Epoch 6/20
160/160 [==============================] - 11s 67ms/step - loss: 1.5399 - val_loss: 1.2820
Epoch 7/20
160/160 [==============================] - 11s 67ms/step - loss: 1.4597 - val_loss: 1.1885
Epoch 8/20
160/160 [==============================] - 11s 67ms/step - loss: 1.3948 - val_loss: 1.1632
Epoch 9/20
160/160 [==============================] - 11s 68ms/step - loss: 1.3582 - val_loss: 1.0866
Epoch 10/20
160/160 [==============================] - 11s 69ms/step - loss: 1.3005 - val_loss: 1.0423
Epoch 11/20
160/160 [==============================] - 11s 69ms/step - loss: 1.2400 - val_loss: 1.0196
Epoch 12/20
160/160 [==============================] - 11s 68ms/step - loss: 1.1775 - val_loss: 1.0293
Epoch 13/20
160/160 [==============================] - 11s 68ms/step - loss: 1.1409 - val_loss: 0.9694
Epoch 14/20
```

Figure 5.1a: Training Encoder-Decoder model



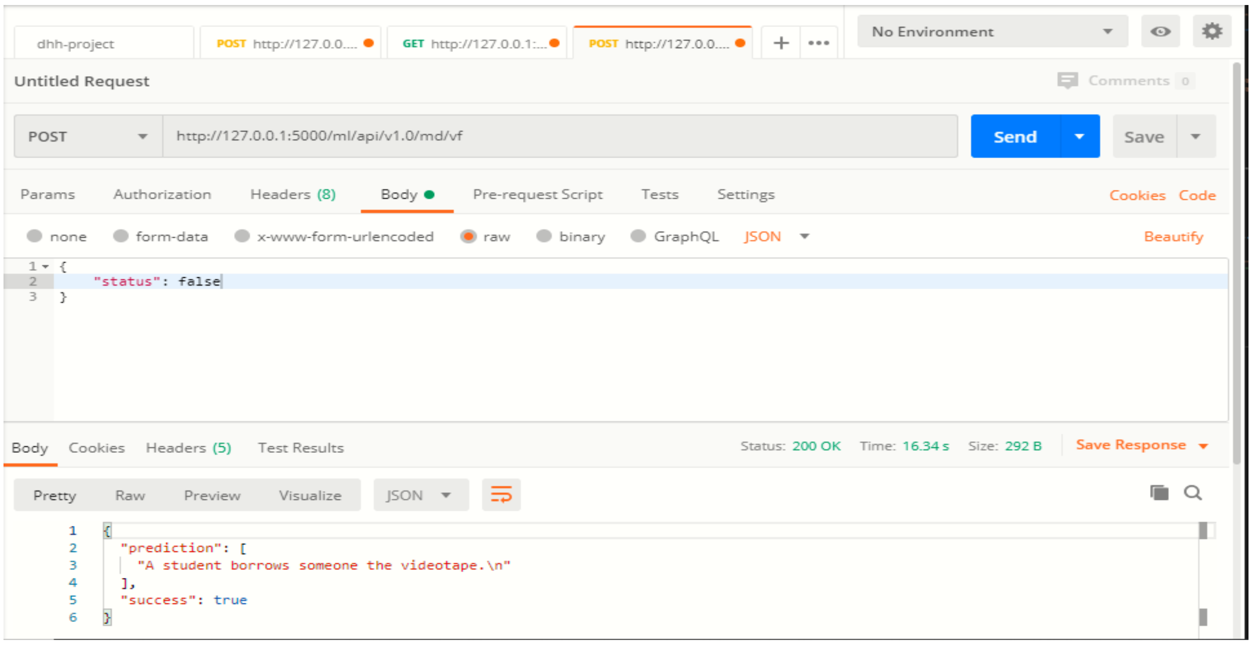
```
160/160 [==============================] - 11s 67ms/step - loss: 1.3948 - val_loss: 1.1632
Epoch 9/20
160/160 [==============================] - 11s 68ms/step - loss: 1.3582 - val_loss: 1.0866
Epoch 10/20
160/160 [==============================] - 11s 69ms/step - loss: 1.3005 - val_loss: 1.0423
Epoch 11/20
160/160 [==============================] - 11s 69ms/step - loss: 1.2400 - val_loss: 1.0196
Epoch 12/20
160/160 [==============================] - 11s 68ms/step - loss: 1.1775 - val_loss: 1.0293
Epoch 13/20
160/160 [==============================] - 11s 68ms/step - loss: 1.1409 - val_loss: 0.9694
Epoch 14/20
160/160 [==============================] - 11s 68ms/step - loss: 1.1252 - val_loss: 0.9250
Epoch 15/20
160/160 [==============================] - 11s 69ms/step - loss: 1.0703 - val_loss: 0.9123
Epoch 16/20
160/160 [==============================] - 11s 68ms/step - loss: 1.0502 - val_loss: 0.8977
Epoch 17/20
160/160 [==============================] - 11s 67ms/step - loss: 1.0210 - val_loss: 0.8948
Epoch 18/20
160/160 [==============================] - 11s 68ms/step - loss: 1.0362 - val_loss: 0.8848
Epoch 19/20
160/160 [==============================] - 11s 67ms/step - loss: 0.9622 - val_loss: 0.8877
Epoch 20/20
160/160 [==============================] - 11s 68ms/step - loss: 0.9442 - val_loss: 0.8668
Saving model....
```

Figure 5.1b: Training and saving Encoder-Decoder model to file

(ii) Web Service (API)

The Web Service implemented two main endpoints to support the functional requirements of the system. These included a video streaming endpoint and the model selection endpoint. The video streaming endpoint, */ml/api/v1.0/md/vf*, expected a post request with JSON data indicating whether to either returns video frames from the user webcam or call the prediction function to compute the prediction as a JSON response. On the other hand, the model selection endpoint, */ml/api/v1.0/md/<int:model_id>*, receives a GET request and loads the corresponding convolutional neural network such as MobileNet or InceptionV3, as specified in the URL address.

To test whether the various endpoints were working, the Postman application was employed to interact with the web service. Postman ensures that the request and response are in the standard format, and the various HTTP methods such as POST and GET are also supported.



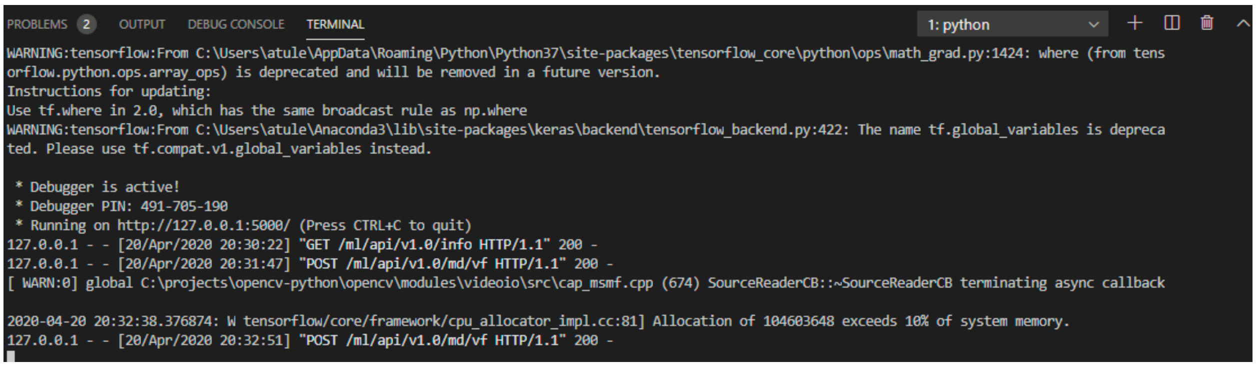
POST http://127.0.0.1:5000/ml/api/v1.0/md/vf

```
{
    "status": false
}
```

Status: 200 OK Time: 16.34 s Size: 292 B

```
{
    "prediction": [
      "A student borrows someone the videotape.\n"
    ],
    "success": true
}
```

Figure 5.2: Model prediction in Postman



```
WARNING:tensorflow:From C:\Users\atule\AppData\Roaming\Python\Python37\site-packages\tensorflow_core\python\ops\math_grad.py:1424: where (from tens
orflow.python.ops.array_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
WARNING:tensorflow:From C:\Users\atule\Anaconda3\lib\site-packages\keras\backend\tensorflow_backend.py:422: The name tf.global_variables is depreca
ted. Please use tf.compat.v1.global_variables instead.

 * Debugger is active!
 * Debugger PIN: 491-705-190
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [20/Apr/2020 20:30:22] "GET /ml/api/v1.0/info HTTP/1.1" 200 -
127.0.0.1 - - [20/Apr/2020 20:31:47] "POST /ml/api/v1.0/md/vf HTTP/1.1" 200 -
[ WARN:0] global C:\projects\opencv-python\opencv\modules\videoio\src\cap_msmf.cpp (674) SourceReaderCB::~SourceReaderCB terminating async callback

2020-04-20 20:32:38.376874: W tensorflow/core/framework/cpu_allocator_impl.cc:81] Allocation of 104603648 exceeds 10% of system memory.
127.0.0.1 - - [20/Apr/2020 20:32:51] "POST /ml/api/v1.0/md/vf HTTP/1.1" 200 -
```

Figure 5.3: Web service running in visual studio code

(iii) Data Processing Pipeline

The Data Processing Pipeline is expected to receive video frames, crop them into an appropriate size depending on the CNN model that is selected, either MobileNet or InceptionV3 model. This component is tested to ensure that the target user in the video frames must not be cropped out. In order to test this, the final output of component, the cropped frames, are visualized to make sure the target user is still visible in the frames, as shown below:

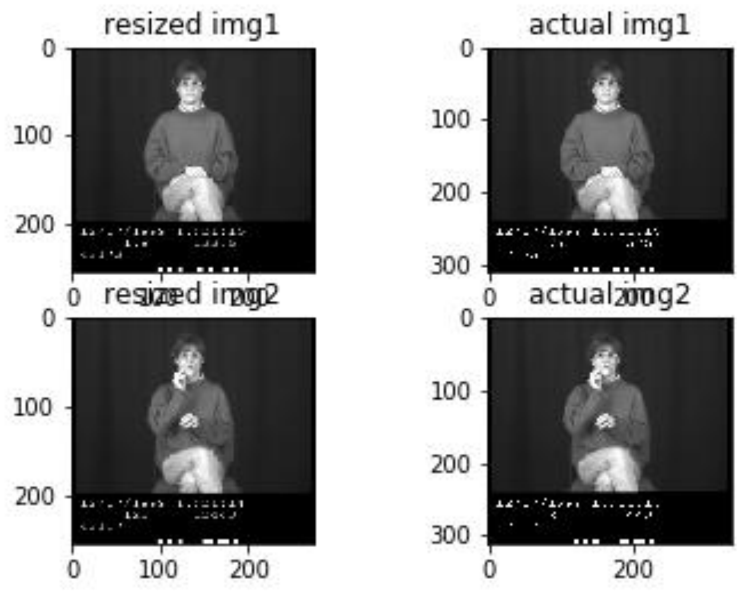
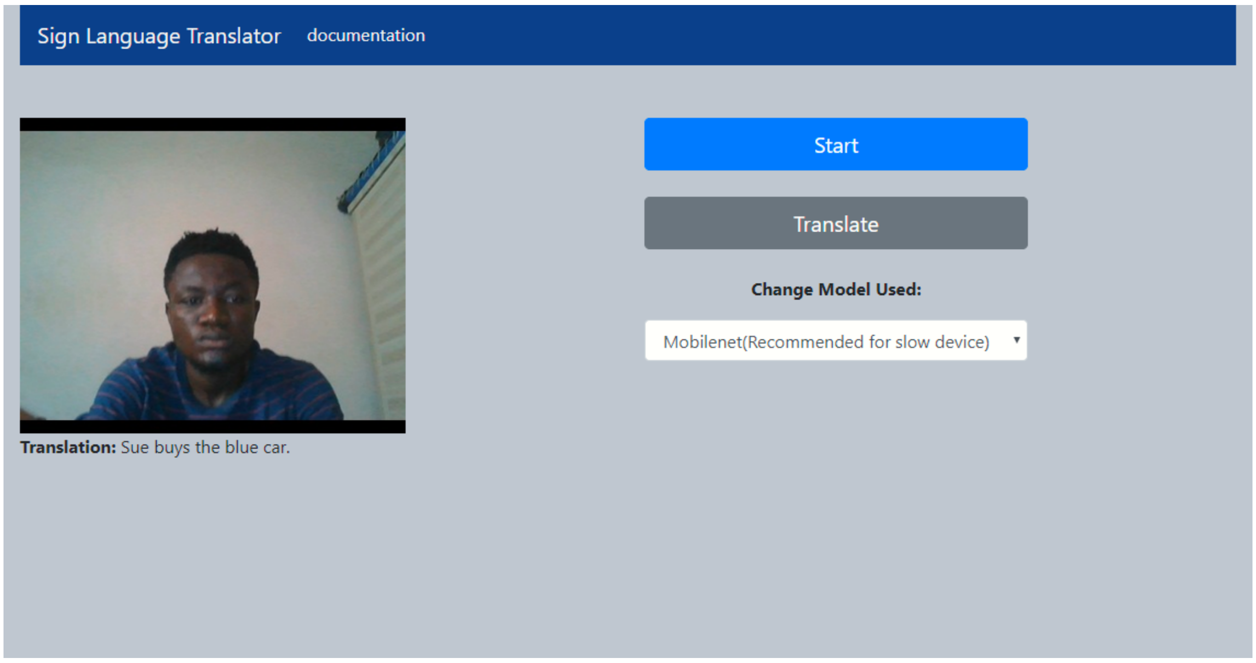


Figure 5.4: Unprocessed and processed frames data

(iv) Web Application Interface

This component was built with mock data to ensure that the various elements of the interface, such as the video streaming section, the start, and translate buttons as well as the model selection form were all in place. The video streaming section was initially replaced with static picture frames.

The start and the translate button were expected to send a post request to a default route when submitted, while the select element sent a get request when selected.



Translation: Sue buys the blue car.

Figure 5.5: Web application interface

5.2 System-level Testing

This testing stage validates the complete and fully integrated system. In this testing technique, all the individual components are harmonized and tested end-to-end to meet the entire system specification. At this stage, the mock data on the web application interface was replaced with dynamic data generated from the backend logic of the system. The generated video frames from the webcam where retrieved and replaced on the web interface to simulate the streaming process. Furthermore, the start and translate buttons are configured to send a post request to the video streaming endpoint to trigger recording, and prediction, respectively when triggered. The select element is also configured to send a GET request to the model selection endpoint, thereby loading one of the two CNN models at the backend.

The Translation Model is trained and saved to a local directory, where the web service is pointed to load and use it for prediction without on-device training. The various endpoints encapsulate the underlying implementation of the Translation Engine, ensuring that different pre-trained models can be easily integrated.

To test the system end-to-end, a user was made to sign using the web application interface, and the predicted results displayed on the application.

5.3 User Testing

In the User Testing stage, the entire system is deployed and made available to users to interact with in order to discover design flaws. This stage ensures that the capacity of the system is measured and can withstand the volume of user requests during production. Furthermore, the ease of use is measured and ensures the system meets its intended purpose.

To satisfy the scope of this project within the limited time available, and due to the requirements of users to understand American Sign Language, only a few test users met this requirement and were invited to test out the application.

The application was deployed to a server, and the URL of the application sent out to the users who were knowledgeable in communicating in the American Sign Language. The particular domain of conversation on which the training data was generated is also made know to the users, so that they do not sign out of context. Their feedback is then collated with regards to the accuracy of the model, and the intuitive nature of the web interface.

5.4 Analysis of Test Results and Challenges

The user testing was not very successful due to the lack of knowledge in the American Sign Language by most users and the author as well. As a result, only a few sign language communications were learned by the author and then used to test the application. Users found the web application interface intuitive and simple to use.

However, even though the web service was fully implemented, there was difficulty in retrieving video frames from the server with Angular HttpClient, this led to only fully implementing the web application without the accompanying mobile platform. The mobile interface will be implemented in the future iteration of this project. Other challenges were developing proper performance metrics to evaluate the Translation Model performance, and therefore only the empirical performance was used. And finally, due to the limited computing power available, the model was not trained for several epochs, and only small training data was employed to train the model as well. There was be a lot of improvement in model performance if a lot of training data was used.

Chapter 6: Conclusion and Recommendations

6.1 Summary

The larger goal of this project is to provide a fast and reliable sign language translation that will enhance communication among DHH individuals and those without hearing impairment. This will serve to bridge the lack of accessible facilities for people with DHH challenges in an educational and public institution and help reduce the need for Interpreters. The main goals of this project, as stated from the onset, were achieved:

(i) A novel deep learning architecture 3D-RCNN was employed to achieve faster training time and efficient use of computer memory.

(ii) An end-to-end deep learning model for translating sign language to English within near real-time was implemented.

(iii) The Translation Model was further implemented as a web service and deployed for public accessibility.

(iv) A web-based application interface that enabled users to translate sign language communication English was also implemented and deployed.

(v) And finally, a data pipeline for generating large datasets using YouTube sign language playlists for continuous training of the deep learning model was also implemented.

The system implemented in this project could be adopted and trained to translate any version of Sign Language into English or any target language. It can also be easily integrated into other projects as the web service, or source code of the Translation model is made available. Furthermore, the Encoder-Decoder model, as implemented in this project, could also be modified to do Language Translation. All this achieved with a lightweight Encoder-Decoder architecture model and the use of Transfer Learning to reduce the computing power requirements of traditional machine learning models.

6.2 Major Limitations

However, this system is without its challenges. As with any machine learning model, there is an inherent bias in the model due to the characteristics of the generated dataset. The YouTube playlist that was employed to create the training data was produced by a single female author. This leads to a bias of the model towards female personalities. This is likely expected, but this challenge could be avoided by producing dataset that reflects both genders. Furthermore, as a conversational model, the generated dataset needs to reflect all domains that the model is going to be used in. One approach could be to develop a specific model for specific conversational contexts. Another method could mean collecting enough data to train a general model that could be used in any conversational domain. In this version of the implemented model, specific data was generated to reflect trivial greetings, basic personal introduction, and Christmas season’s conversations, among others. Hence, this model would not work correctly when sign language communication is outside the domain the dataset was generated.

Furthermore, there is limited computing power needed to train the model on the large dataset that is generated. Paid cloud services could be explored in the future to train the model on a larger dataset to increase the predictive capacity of the model.

6.3 Other Limitations

Other limitations include the inability to build a mobile platform for the system. A mobile application will increase the number of people who would use the system and make the system more beneficial during conversations. However, fully implementing a web service and web application proofs that it is possible to build a mobile application for this system. Future iterations of this project will mitigate this limitation. Furthermore, the web application does not currently look appealing. More design concepts will be infused into it in future work.

6.4 Recommendations and Future Works

Several improvements can be made to the existing system. A major one could involve experimenting with the size of the Encoder-Decoder model, to figure out the best network size that improves the general performance of the Translation system. However, this will require large computing power to quickly try out different model parameters. It was not possible to try different parameters due to the limited nature of the available computing power. Furthermore, more time could be used to manual sort through the dataset that is automatically generated with the data pipeline. This will help improve the quality of the generated dataset, thereby increasing the performance of the Translation model.

It is recommended that varied YouTube playlist be used in generating the dataset to reduce the model bias. This model could also be used to implement Sign Language Translation in any language. This is highly encouraged so that global accessibility could be improved for DHH individuals worldwide.

Beyond the above recommendation, the following will be implemented in future iterations of the project as they are very essential to achieve the goals of this project:

(i) Implement a mobile application interface for both android and IOS operating systems.

(ii) Generate large and varied datasets to implement a general Translation Model for most conversational domains.

References

[1] Alexa Kuenburg, Paul Fellinger, Johannes Fellinger. 2014. Health Care Access Among Deaf People, *The Journal of Deaf Studies and Deaf Education*, Volume 21, Issue 1, January 2016, Pages 1–10, https://doi.org/10.1093/deafed/env042

[2] British Death Council. (2014). Accessing Public Services: Issues for Deaf People. Retrieved from https://bda.org.uk/wpcontent/uploads/2017/03/BDA_Accessing_Public_Services-Issues_for_Deaf_People-London_Boroughs_12-2014.pdf

[3] Edwards Jim. 2013. Scared, abandoned and ignored: public services for deaf people. *The Guardian,* https://www.theguardian.com/public-leaders-network/2013/may/10/public-services-deaf-people-loophole

[4] Gaolin Fang and Wen Gao. 2002. An SRN/HMM system for signer-independent continuous sign language recognition. *Proceedings of Fifth IEEE International Conference on Automatic Face Gesture Recognition*, Washington, DC, USA, 2002, pp. 312-317. doi:10.1109/AFGR.2002.1004172

[5] Jie Huang, Wengang Zhou, Houqiang Li and Weiping Li, "Sign Language Recognition using 3D convolutional neural networks," *2015 IEEE International Conference on Multimedia and Expo (ICME)*, Turin, 2015, pp. 1-6. doi: 10.1109/ICME.2015.7177428

[6] Jin-Woo Chung, Ho-Joon Lee , Jong C. Park. Improving accessibility to web documents for the aurally challenged with sign language animation. *Proceedings of the International Conference on Web Intelligence, Mining and Semantics*, May 25-27, 2011, Sogndal, Norway. Doi:10.1145/1988688.1988727

[7] Jiyong Ma, Wen Gao, Jiangqin Wu and Chunli Wang. 2000. A continuous Chinese sign language recognition system. *Proceedings Fourth IEEE International Conference on Automatic Face and Gesture Recognition (Cat. No. PR00580),* Grenoble, France, 2000, pp. 428-433. Doi: 10.1109/AFGR.2000.840670.

[8] Luft Pamela. 2000. Communication barriers for deaf employees: Needs assessment and problem-solving strategies. *in* work. February 2000. 14(1). pp. 51-59. https://www.researchgate.net/publication/11026132_Communication_barriers_for_deaf_employees_Needs_assessment_and_problem-solving_strategies.

[9] Matt Huenerfauth. 2009. A Linguistically Motivated Model for Speed and Pausing in Animations of American Sign Language. Proceedings ACM Transactions on Accessible Computing (TACCESS), Volume 2 Issue 2, June 2009. Doi: 10.1145/1530064.1530067.

[10] World Health Organization. 2019. Deafness and Hearing Loss. https://www.who.int/news-room/fact-sheets/detail/deafness-and-hearing-loss

[11] Syed A. Mehdi, and Y. N. Khan. (2002). Sign language recognition using sensor gloves. *Proceedings of the 9th International Conference on Neural Information Processing, 2002. ICONIP' 02.*, Singapore, 2002, pp. 2204-2206 vol.5. doi: 10.1109/ICONIP.2002.1201884

[12] Stephen Cox, Michael Lincoln, Judy Tryggvason, Melanie Nakisa, Mark Wells, Marcus Tutt, Sanja Abbott. Tessa, a system to aid communication with deaf people. *Proceedings of the fifth international ACM conference on Assistive technologies,* Edinburgh, Scotland, pp. 205 – 212, July 2002. Doi:10.1145/638249.638287

[13] Eric P. Tudzi, John Bugri, and Anthony Danso. 2017. Towards Accessible Built Environments in Universities in Ghana: An Approach to Inclusiveness Assessment.

Disability, CBR, and Inclusive Development (DCID) Journal, 28(1), 189-206. doi: https://doi.org/10.5463/dcid.v28i1.592.

[14] Carlos M. Travieso, Jesus B. Alonso and Miguel A. Ferrer. Sign language to text by SVM. *Seventh International Symposium on Signal Processing and Its Applications, 2003. Proceedings.*, Paris, France, 2003, pp. 435-438 vol.2. doi: 10.1109/ISSPA.2003.1224907

[15] D. Uebersax, J. Gall, M. Van den Bergh and L. Van Gool. Real-time sign language letter and word recognition from depth data. *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, Barcelona, 2011, pp. 383-390. doi: 10.1109/ICCVW.2011.6130267.

[16] T. Shanableh, K. Assaleh and M. Al-Rousan. Spatio-Temporal Feature-Extraction Techniques for Isolated Gesture Recognition in Arabic Sign Language. in *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 37, no. 3, pp. 641-650, June 2007. Dosi:10.1109/TSMCB.2006.889630.