Gesture Language Translator

Documentation

December 8, 2021

1 Product Description

The product is a system for recognizing sign language gestures from the ASL alphabet and converting it to text, then to speech. This solution could be useful in a way that individuals who are only able to communicate via sign language could be understood by people who are not familiar with it. Below, the main parts of the project are discussed, such as gesture recognition, word formation text-to-speech, GUI, etc.

2 Gesture recognition

Gesture recognition is one of the backbones of the application. The main idea of this block is for the system to be able to recognize hand gestures and return the appropriate character as a prediction.

For this a model was created a with the help of an open-source tool called MediaPipe. MediaPipe Hands can track 21 landmarks, and return their coordinates on the image frame. The model uses labeled landmark coordinate data extracted from hand signals by MediaPipe. This data is used to train a classification model, that is a neural network with the following structure:

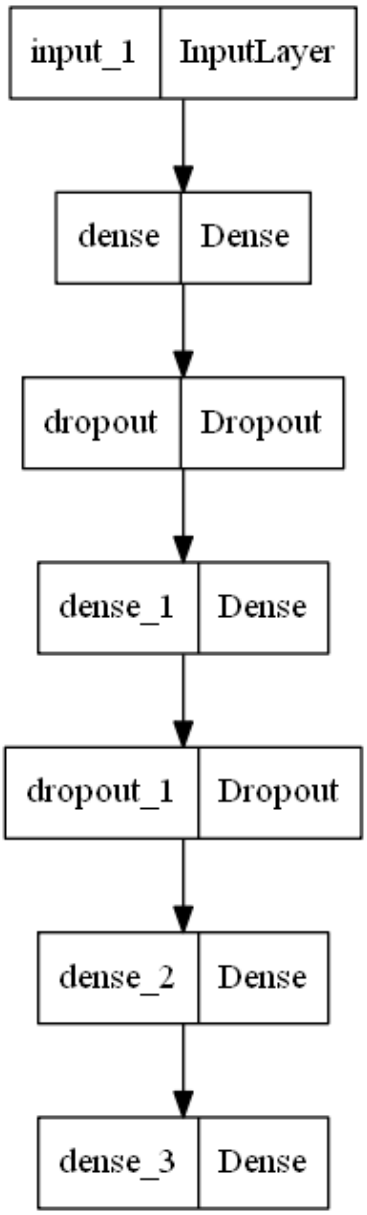

Figure 1: Structure of model

The 27 possible classes are the 26 letters in the English alphabet plus a 'space' symbol to separate words. When the training was finished, the model showed a high validation accuracy of around 94%, proving that the model was ready to be used for the rest of the project.

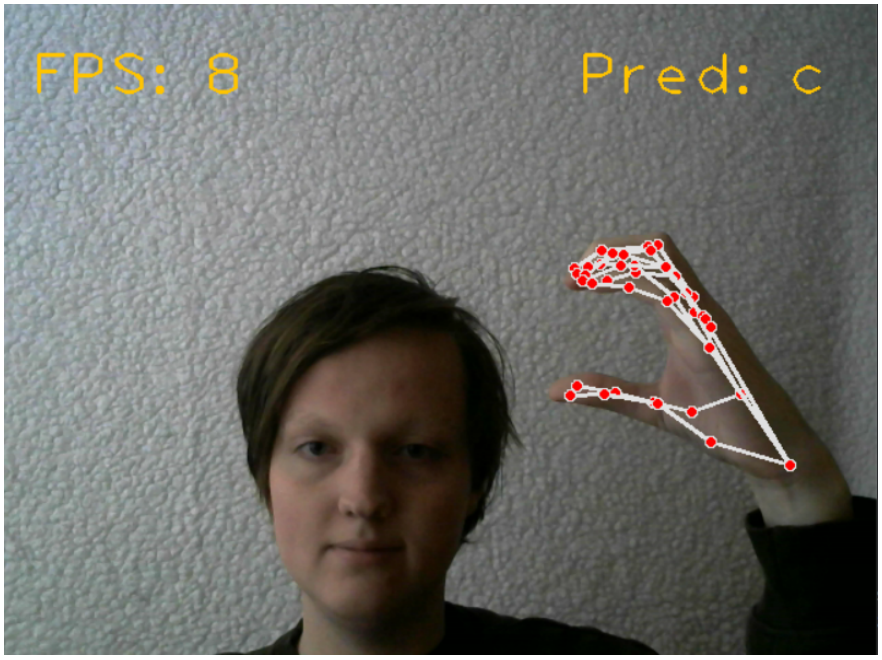

Figure 2: Working model, prediction in top right corner

3 Word formation

When using the model in practice, it can predict a letter multiple times even if the user only meant to show it once. To avoid this, it was implemented that there must be a 'none' symbol recognized by the model before reading a new letter as an input. The application uses the 'space' characters to separate words. Auto correct is the essential part of forming words from the classified letters. Even with good accuracy, model can miss-classify or user can get the gestures wrong, having auto correct feature adds confidence to the output. Here, we have used python module 'autocorrect', which compares the predicted word with the dictionary and corrects it to a sensible word. Built-in function 'Spell' is called on the string, and outputs the corrected spelling.

4 Text-to-speech

An important feature of the solution is the text to speech part, as this will convert the recognized words into speech that can be heard by another party. For this the application uses Google's open source Text-to-Speech model, GTTS. The solution uses GTTS for translating prediction from the model which classifies the gesture and returns the text. Here, "gtts.gTTs()" function was used which takes the text and language as input, and returns an audio output. Then the audio is saved temporarily, and played before deletion.

5 Software Architecture

We use client-server architecture, in which the server hosts, delivers, and manages most of the resources and services to be consumed by the client. This type of architecture has one or more client computers connected to a central server over internet. We used Google Colab as a backend server, and we developed a desktop GUI application using python.

On the server, we have the trained model so basically we receive the image from the user and predict the letter. Then, we return the result as JSON to the user.

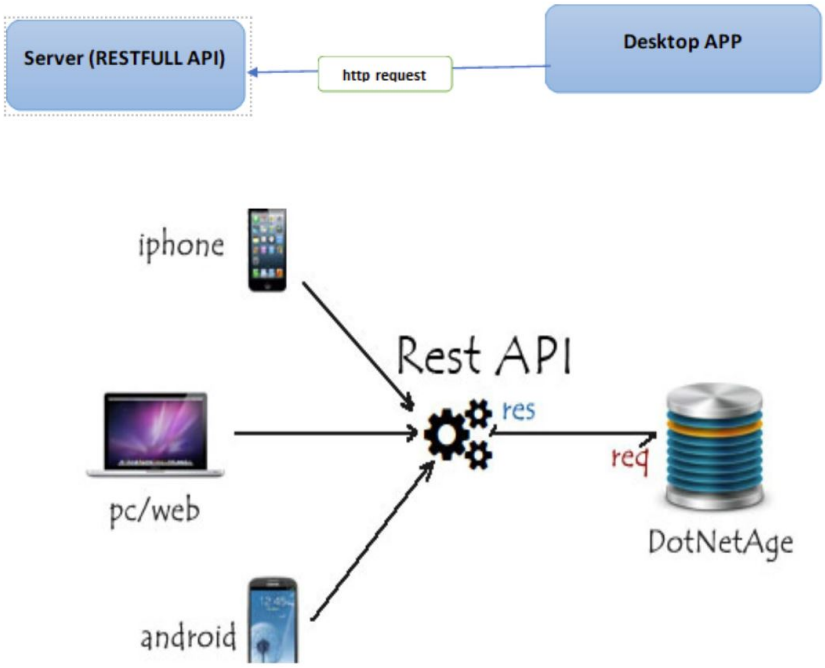

Figure 3: Software Architecture

And here is the documentation of the API:

API Link	Request type	Request body	Expected response
/HandDetectByPhoto	Post	{“file”:your_Image}	{“Class”:A}
/HandDetectByVideo	Post	{“file”:your_Video}	{“class”:I am Mohamad}

Figure 4: API Table

5.1 Testing

We have done the following types of testing:

- Unit Testing:

Here we have written a python script for each function in our app, with specific parameters. Then we compare the result, and if it matches the expected one that means the test is passed.

- API Testing:

  Here we send requests to our API and compare the json response. If it matches the expected one then the test is passed.

- Stress Testing:

  Here we have written a flask-app which simulates the users' behavior on our server. We send many requests at the same time and check the server response and when it could crash.

We run this testing unit when we add, edit or delete any features of our app to check that the app works well and is not affected by the changes.

5.2 Tracking system

When any user calls any service of our API services we store the following data in a log file:

- Name of the service.
- Date of service call.
- Input Parameters of the service.
- The output of the service.
- Execution time.

5.3 Auto Deployment

Since we use Google Colab as a backend Server, and we have not found any tool or service to do such automation deployment to Google Colab, we wrote a python script which will upload the API to Google Drive and from Google Colab we can integrate it with Google Drive.

6 GUI

The GUI has many important parts. First, there is a feedback screen where the user can see what the camera senses, thus being able to move their hand into frame. It also shows the prediction for the current signal shown by the user, and the confidence of said prediction. As seen on the picture below, there are several buttons with different functions on the GUI.

The GUI was designed using the PySimpleGUI library. PySimpleGUI is a python library that wraps Tkinter, Qt (pyside2), wxPython and Remi (for browser support), allowing very fast and simple-to-learn GUI programming. All project functions are included in the GUI to be easily experienced:

- The real-time sign language translating are shown in the 'I am gesturing' sub-window.
- The 'I am saying' sub-window display the output text and voice (using the 'LISTEN' button).
- The outputs are corrected by the auto-correction function after every 'SPACE' gesture.

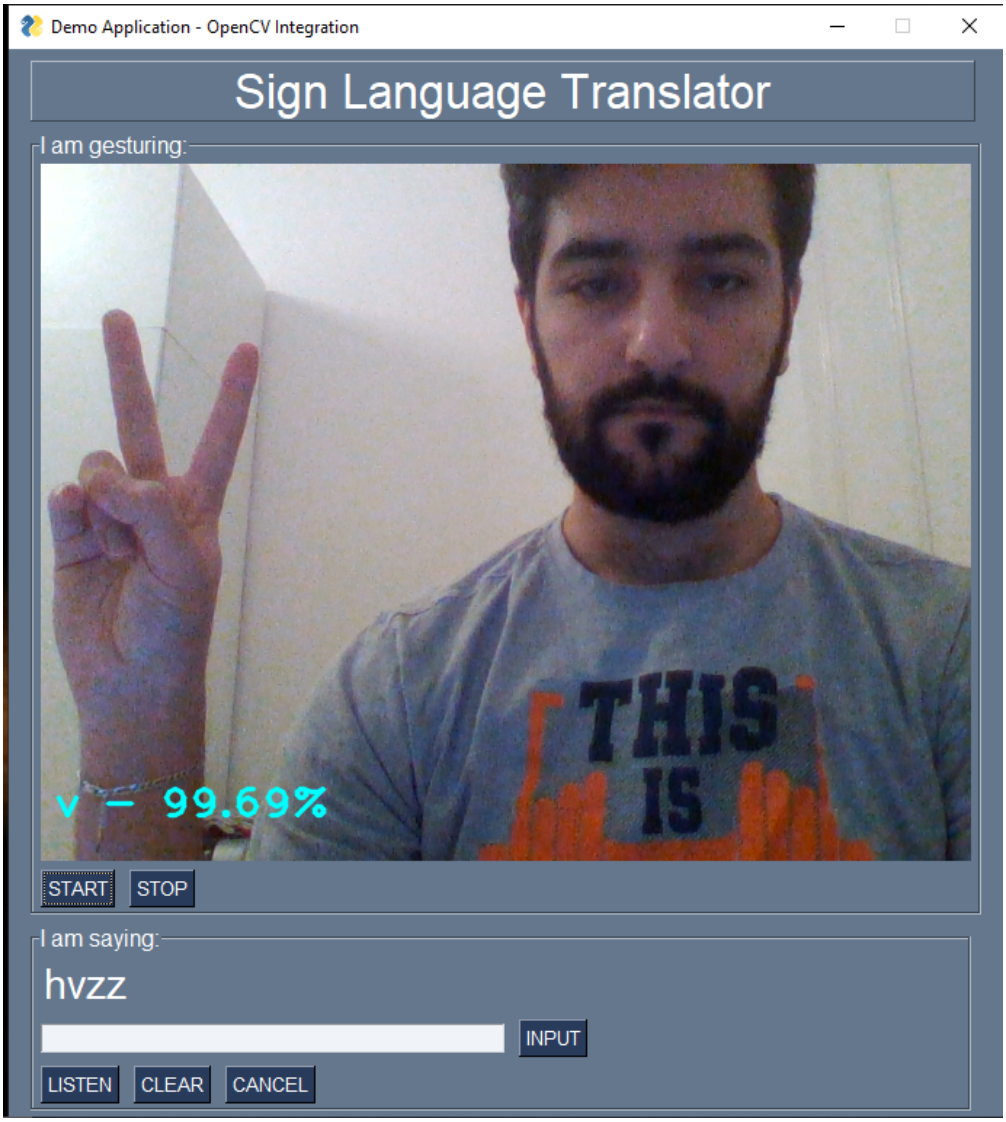

Figure 5: Image of the GUI