SIGN LANGUAGE TRANSLATOR USING IMAGE DETECTION

FOUNDATIONS OF AI (CSE1014)
FINAL REPORT

SUBMITTED TO: PROF. VIJAYALAKSHMI A

Project done by:
Sarthak Vasal - 19BAI1050
Hasnain Sikora - 19BAI1072

ABSTRACT

Our project aims to help the deaf and the mute who constitute a little more than 1% of the Indian population according to a 2013 consensus which tracks persons with disabilities. With an increasing number of people needing basic ASL in their day to day lives, only 250 certified sign language interpreters function throughout the nation. The goal of the final product is to provide an easy and accessible system for anyone around the globe to interpret and translate ASL in real time.

INTRODUCTION

AS PER THE 2011 INDIAN CENSUS THERE ARE 1.3 MILLION PEOPLE WITH HEARING IMPAIRMENT

Our project aims to create a simple image detection system which tracks hand gestures and thus recognizes the signs. These signs are detected using a self made dataset.
The project is divided into 2 components, namely letter detection and phrase detection. The standardized ASL (American Sign Language) hand signs for all 26 letters of the English alphabet and a few general phrases such as 'hello', 'thankyou', 'goodbye' etc.
The only requirements to use the software is the need for a primary camera or a web cam that can assess the real time video input and detect the hand signs using the pre trained AI.
The implementation of the AI was carried out with the help of the following tools

- TensorFlow
- TensorFlow Model Garden
- Protocol Buffers
- COCO API
- LabalImg

The AI uses the following models:

- SSD MobileNet V2 FPNLite 640x640
- SSD MobileNet V2 FPNLite 320x320

The SSD Mobilenet V2 Object detection model with FPN-lite feature extractor, shared box predictor, and focal loss, trained on COCO 2017 dataset with training images scaled to 640x640 and 320x320 respectively.
Once these signs are detected they are translated to speech and text.
The outcome of this project will not only help the deaf and the mute but also the ones around them who would be able to understand them.

RELATED WORKS

BLAZEPALM BY GOOGLE AI (BLOG)

This is from Google's mediapipe paper. Implemented in MediaPipe, it is an open source cross platform framework for building pipelines to process perceptual data of different modalities, such as video and audio. This approach provides high-fidelity hand and finger tracking by employing machine learning (ML) to infer 21 3D keypoints of a hand from just a single frame.

How we stand out:

> The documentation only includes the framework of the hand detection using AI. The software is limited to the handful of phrases pretrained into the system. It doesn't cater to the needs of the deaf and mute.

LINGOJAM

A free online software that translates text and displays illustrations of hand signs as output using the ASL dataset.

How we stand out:

> LingoJam is a one way translator, it only translates English text to ASL and not vice versa. There is no audio interface. Translates letter by letter, doesn't include phrases.

HAND TALK TRANSLATOR

Hand Talk app automatically translates text and audio to American Sign Language (ASL) and Brazilian Sign Language (Libras) through artificial Intelligence. It uses a customizable avatar that acts out the hand sign with a 360° view.

How we stand out:

> The hand talk translator app only uses text as an input. It lacks real time hand sign translation from ASL to English. The app is specific only for ASL letters and not phrases.

TOOLS USED

TENSORFLOW

A free and open-source software library for machine learning. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks.

TENSORFLOW MODEL GARDEN

A repository with a number of different implementations of state-of-the-art models and modeling solutions for TensorFlow.

PROTOCOL BUFFERS AKA PROTOBUF

Google's language-neutral, platform-neutral, extensible mechanism for serializing structured data.

COCO API

A large image dataset designed for object detection, segmentation, person keypoints detection, stuff segmentation, and caption generation.

LABELIMG

A graphical image annotation tool. are saved as XML files in PASCAL VOC format. It is written in Python and uses Qt for its graphical interface.

METHODOLOGY

The workspace folder consists of the following files and is organized as given in the TF2 API documentation:

ANNOTATIONS

This folder will be used to store all *.PBTXT files and the respective TensorFlow *.record files, which contain the list of annotations for our dataset images.

MODELS

This folder will contain a sub-folder for each of our custom models whose config file is tweaked as per our project. Each subfolder will contain the training pipeline configuration file *.config, as well as all files generated during the training and evaluation of our model.

IMAGES

This folder contains a copy of all the images in our dataset, as well as the respective *.xml files produced for each one, once labelImg is used to annotate objects.

images/train: This folder contains a copy of all images, and the respective *.xml files, which will be used to train our model.

images/test: This folder contains a copy of all images, and the respective *.xml files, which will be used to test our model.

YOLO OBJECT DETECTION

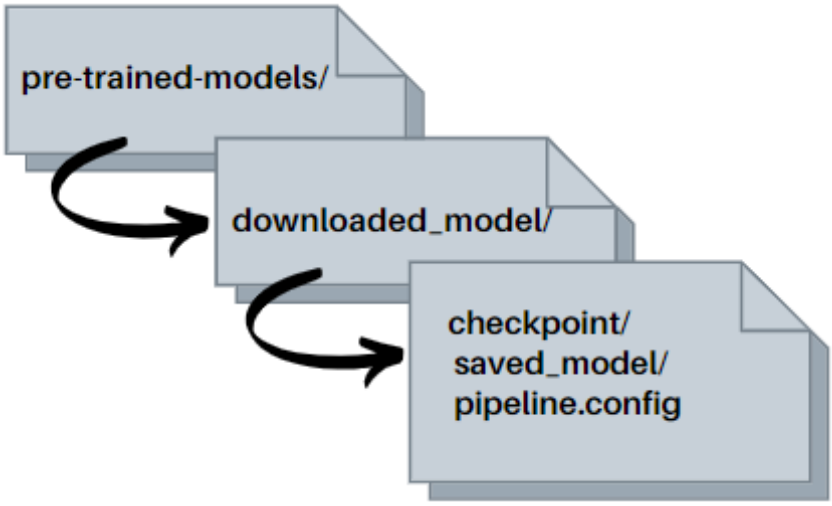
YOLO is an abbreviation for the term 'You Only Look Once'. This is an algorithm that detects and recognizes various objects in a picture (in real-time). Object detection in YOLO is done as a regression problem and provides the class probabilities of the detected images.

PRE-TRAINED-MODELS

This folder will contain the downloaded pre-trained models from the "Tensorflow model zoo", which shall be used as a starting checkpoint for our training jobs.

To initiate our training job - we need to download the pre-trained models that we wish to use. This can be done by simply clicking on the name of the desired model in the table found in TensorFlow 2 Detection Model Zoo. Clicking on the name of your model should initiate a download for a *.tar.gz file.

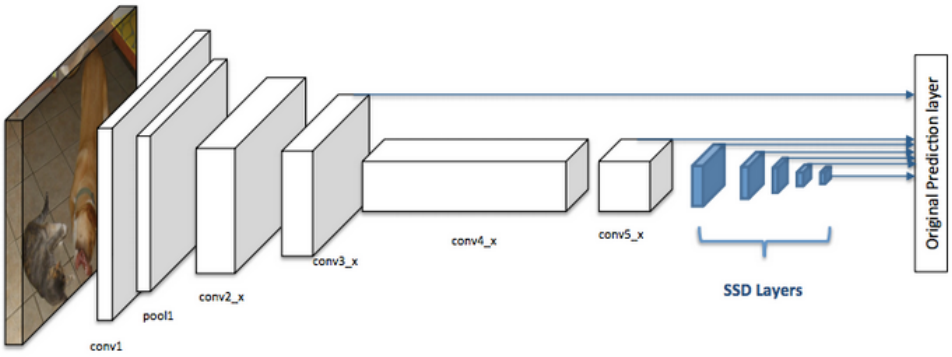
Once the *.tar.gz file has been downloaded, open the *.tar folder that you see when the compressed folder is opened, and extract its contents inside the "pre-trained-models" folder

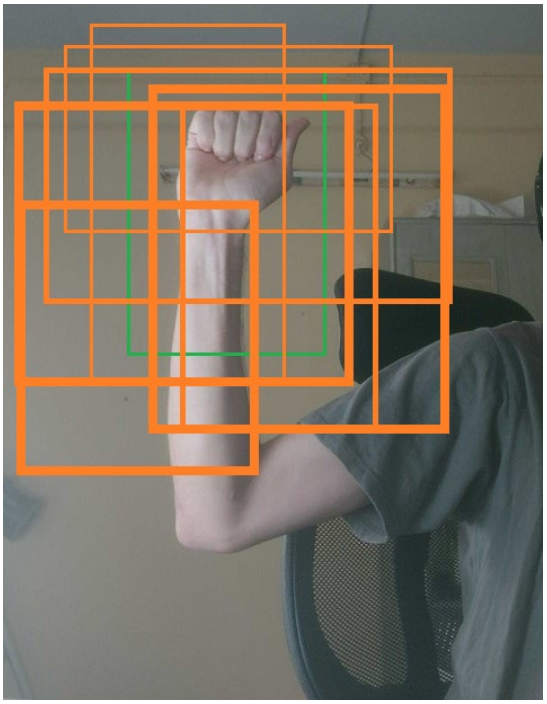


The pipeline.config files is a document with all the default configurations of the pre-trained model which includes the training and testing paths, number of steps, batch sizes, and class size. Our focus is to be directed to editing this config file so as to tweak it to work with our project.

SINGLE SHOT DETECTOR

SSD is a single shot detector- meaning it only needs one shot at the image to analyze and calculate predictions. What it does is discretizes the output space of bounding boxes into a set of default boxes over different aspect ratios and scales per feature map location which in or project's case is MobileNet. These bounding boxes are laid over some ground truth which we have already initialized using labelImg. During prediction time, the network generates scores for the presence of each object category in each default box and produces adjustments to the box to better match the object shape





METHODOLOGY (BRIEF)

Now, for implementing our training models, we have used jupyter notebook and the TensorFlow library. Now, that we've established the importance of SSD's speed and accuracy, we then move on to add it into our project step by step. Since our project deals with sign-language translation for alphabets and phrases – we split our working directories into two.
Our python code consists of basic code to train two of our models, SSD 320X320 and SSD 640x640. As mentioned before these two models will ensure different accuracies by processing the image in different resolutions. In both of our directories we create labels for our alphabets as well as phrases as shown below:

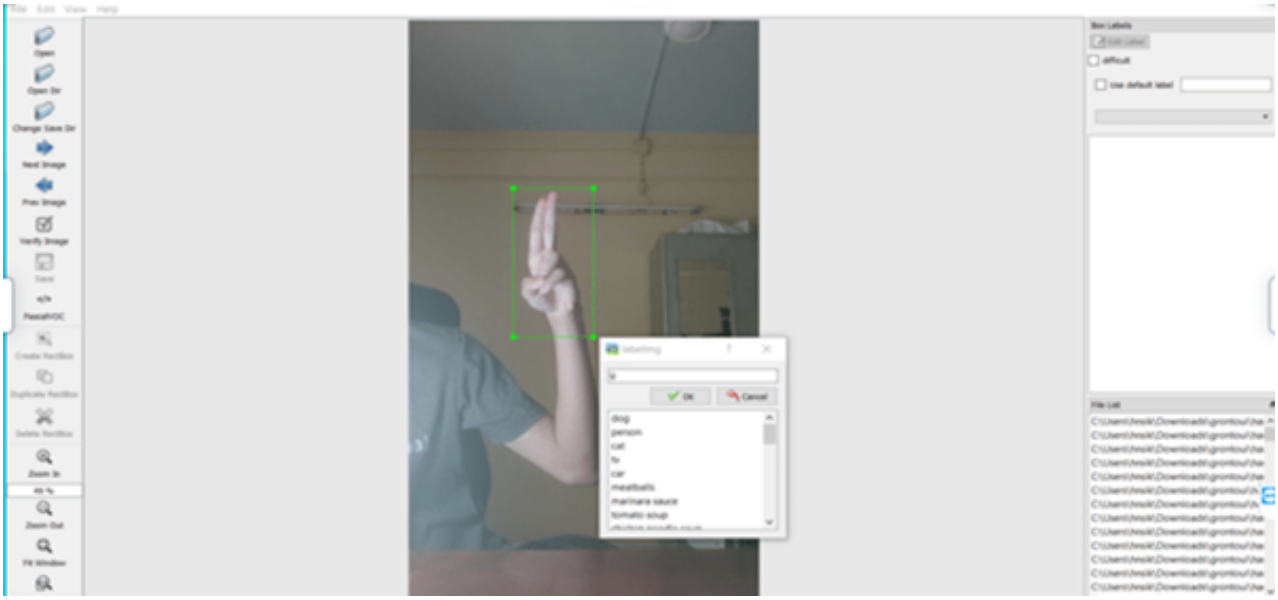
```
labels = [{'name':'A', 'id':1},
          {'name':'B', 'id':2},
          {'name':'C', 'id':3},
          {'name':'D', 'id':4},
          {'name':'E', 'id':5},
          {'name':'F', 'id':6},
          {'name':'G', 'id':7},
          {'name':'H', 'id':8},
          {'name':'I', 'id':9},
          {'name':'J', 'id':10},
          {'name':'K', 'id':11},
          {'name':'L', 'id':12},
          {'name':'M', 'id':13},
          {'name':'N', 'id':14},
          {'name':'O', 'id':15},
          {'name':'P', 'id':16},
          {'name':'Q', 'id':17},
          {'name':'R', 'id':18},
          {'name':'S', 'id':19},
          {'name':'T', 'id':20},
          {'name':'U', 'id':21},
          {'name':'V', 'id':22},
          {'name':'W', 'id':23},
          {'name':'X', 'id':24},
          {'name':'Y', 'id':25},


          ]
```

```
labels = [
    {'name':'Goodbye', 'id':1},
    {'name':'Hello', 'id':2},
    {'name':'Love', 'id':3},
    {'name':'No', 'id':4},
    {'name':'Please', 'id':5},
    {'name':'Sad', 'id':6},
    {'name':'Sorry', 'id':7},
    {'name':'Thanks', 'id':8},
    {'name':'Welcome', 'id':9},
    {'name':'Where', 'id':10},
    {'name':'Yes', 'id':11},
     ]
```

METHODOLOGY

Once we are done with labels, we move into creating ".ptxt" file that stores all of these. This file will be used to access the labels later on.
Before going to the next step, we make sure that all our training and testing images have been properly labelled:



Once we are done labelling our images, we then use the "generatetfrecord.py" file to create TensorFlow records for both our training and testing images. This file ensure that the images our attached into a file with their respective labels and thus is useful when we provide it during our fine-tuning of the pre-trained model.

We, then download and extract our models and finetune it to have the same number of classes as our label class and modify all the paths to TFrecords and the ".pbtxt" file as well as modifying our batch sizes. Once fine-tuning is done, we use python code to run the mainmodel.py to train our models for about 5000 steps.

METHODOLOGY

FRAMEWORK USED

For our final implementation, we have used flask and html to efficiently handle python code and packages alongside with html. We've imported variables which point to the directories that have our custom fine-tuned models as shown below:

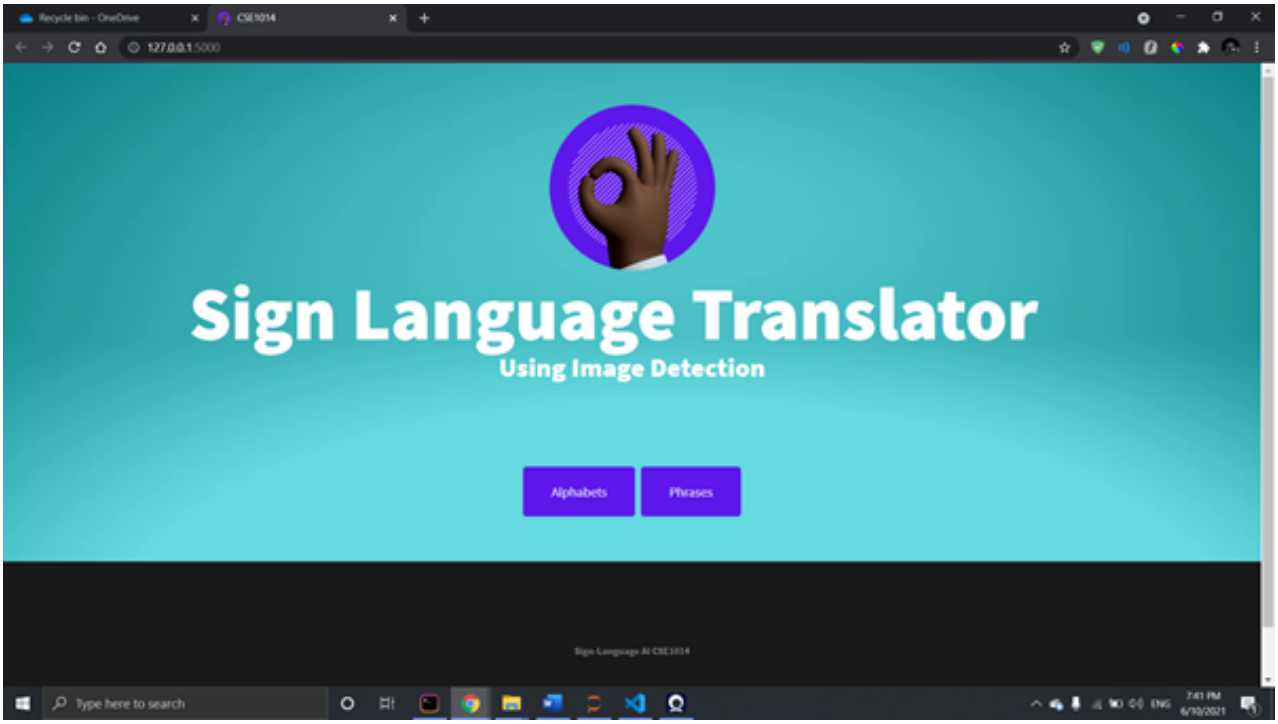
```
config_file = 'C:/Users/hnsik/Pyth/CSE1014/Phrases/Tensorflow/workspace/models/my_ssd_mobnet/pipeline.config'
check = 'C:/Users/hnsik/Pyth/CSE1014/Phrases/Tensorflow/workspace/models/my_ssd_mobnet'
```

We also create functions such as gen_frames() and detect_fn() which will be used to generate an OpenCV frame on the website and visualize our hand's and detect their labels and store them in a list. We will filter out this list to remove any null values and repeated values so that it remains clean. Finally we pass this list into a text to speech function that converts the list into a speech array.

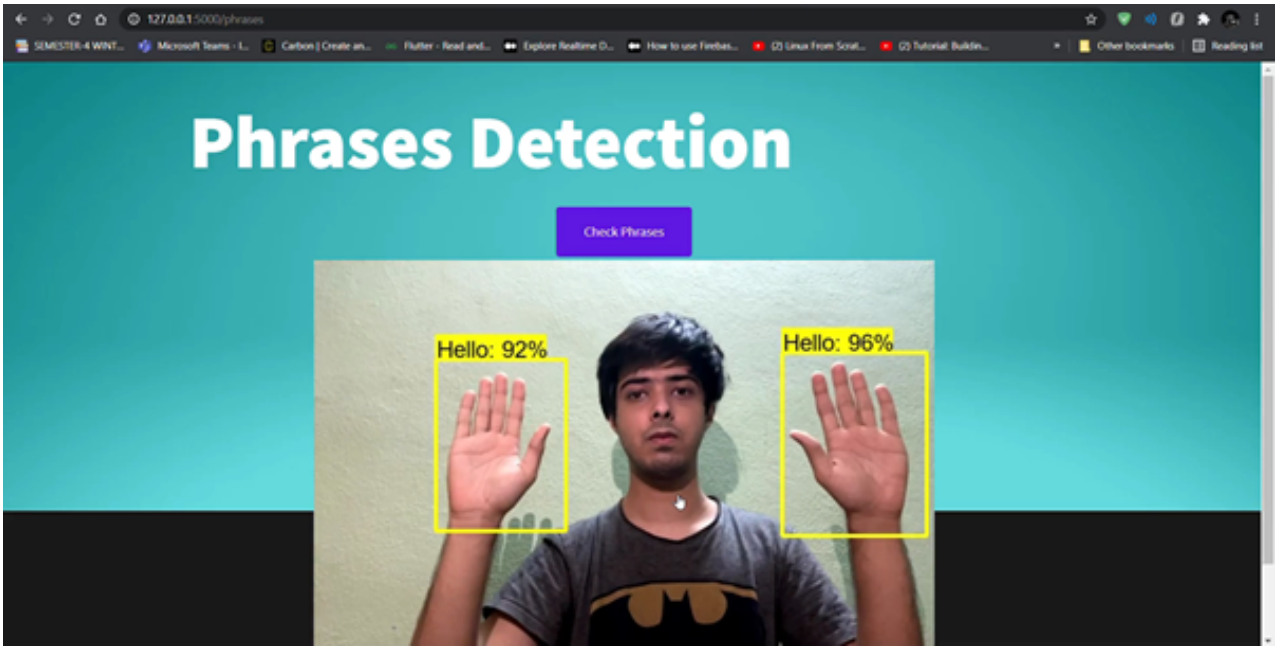
```
def text_to_speech(para):
    time.sleep(2)
    print(para)
    engine = pyttsx3.init()
    voices = engine.getProperty('voices')
    engine.setProperty("rate", 178)
    engine.setProperty('voice', voices[1].id)
    engine.say(para)
    engine.runAndWait()
```

IMPLEMENTATION

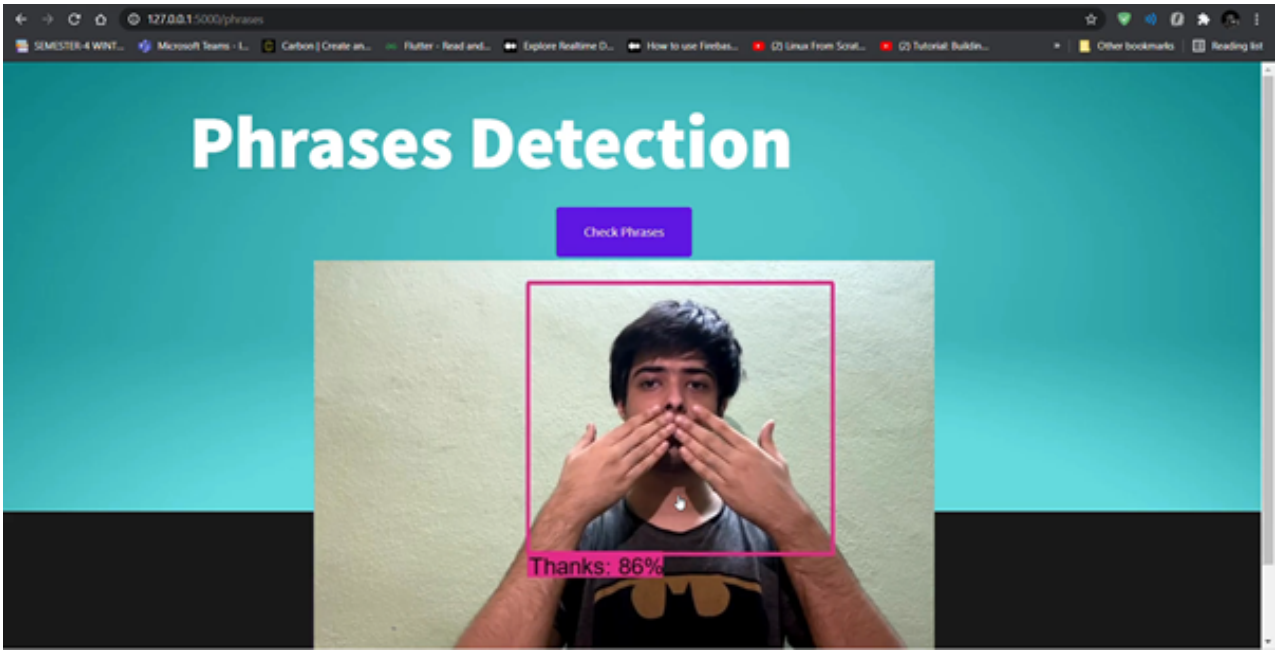
HOME PAGE

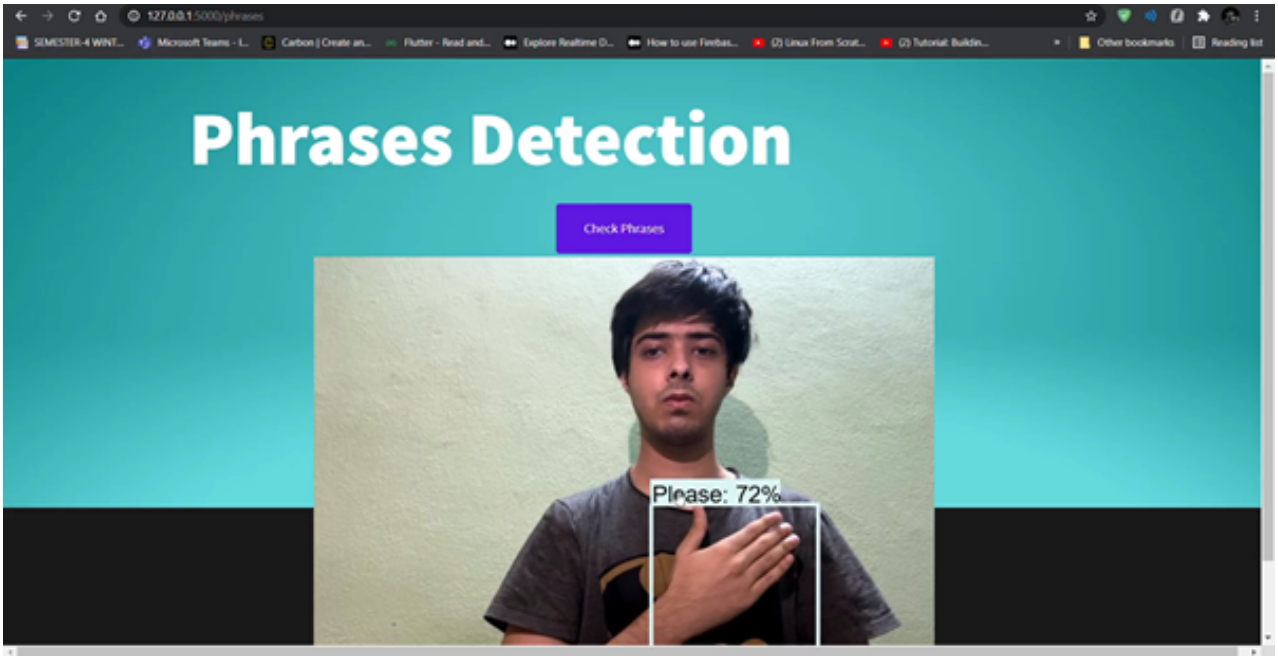


DETECTION PAGE



IMPLEMENTATION



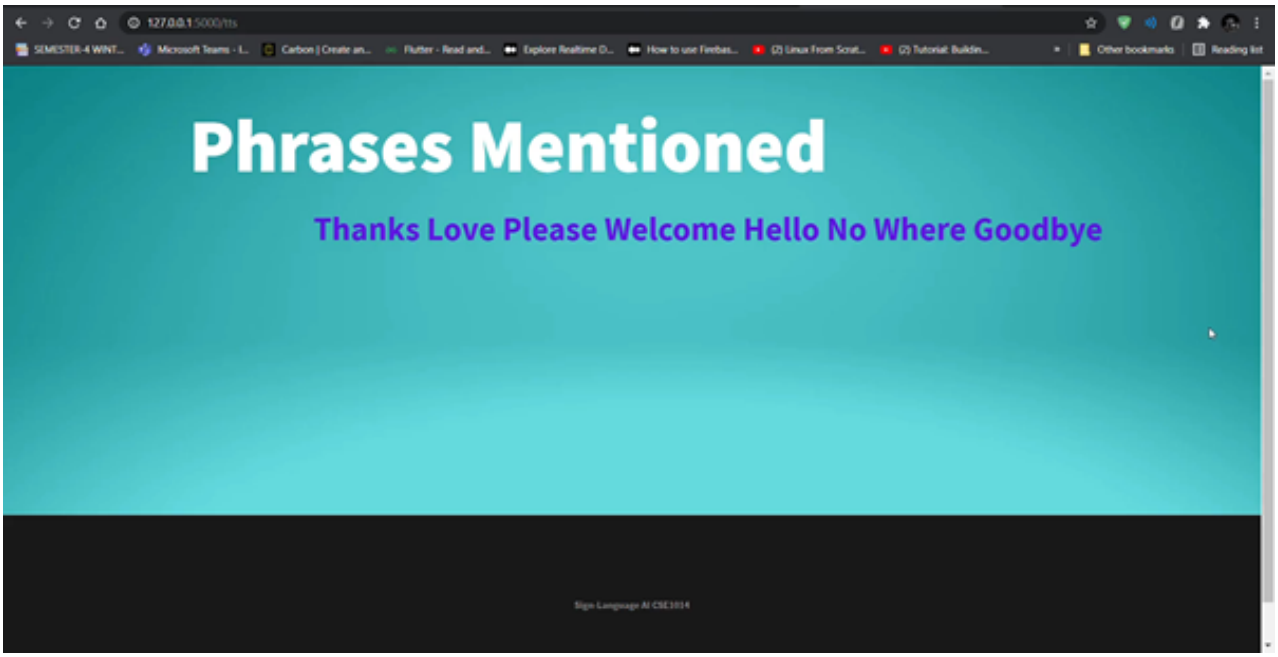


IMPLEMENTATION

TERMINAL LOGS

```
  [[0. 0. 0.]
   [0. 0. 0.]
   [0. 0. 0.]
   ...
   [0. 0. 0.]
   [0. 0. 0.]
   [0. 0. 0.]]], shape=(1, 1080, 1920, 3), dtype=float32)
Num_Detections: 100
Num_Detections: 100
detections['num_detections'] 100
detections['detection_classes'] [ 2  3  2  8  9  2  0  5  9  7  9  9  9  9  9  9  9  5  5  5  0  0  0  0
  0 10  3  0  5  4  0  0  4  4  8  0  5  6  0  5  4  6  6  6  4  0  4  6
  4  5  9  0  9  0  0  9  0  0  9  9  5  0  0  9  4  9 10  0  0  0  9 23
  8  6 23  6  0  9  9 23  9  9 23 23  9  9  9 23  6  8  9  9  9 23  0  9
  4  8  8 10]
detections['num_detections'] 100
detections['detection_classes'] [ 2  3  2  8  9  2  0  5  9  7  9  9  9  9  9  9  9  5  5  5  0  0  0  0
  0 10  3  0  5  4  0  0  4  4  8  0  5  6  0  5  4  6  6  6  4  0  4  6
  4  5  9  0  9  0  0  9  0  0  9  9  5  0  0  9  4  9 10  0  0  0  9 23
  8  6 23  6  0  9  9 23  9  9 23 23  9  9  9 23  6  8  9  9  9 23  0  9
  4  8  8 10]
[]
```

TEXT TO SPEECH PAGE



CODE

AS THE CODE IS TOO LENGTHY TO PASTE, WE'VE PROVIDED A LINK TO THE REPOSITORY DOWN BELOW

THE LINK HAS THREE FOLDERS:

FLASK AI: HAS ALL THE WEBSITE IMPLEMENTATION CODE

ALPHABETS.IPNY: JUPYTER NOTEBOOK FOR ALPHABETS MODEL TRAINING

PHRASES.IPNY: JUPYTER NOTEBOOK FOR PHRASES MODEL TRAINING

https://github.com/hasnain40247/AISignLanguageTranslation/

VIDEO DEMONSTRATION OUTPUT LINK:

https://drive.google.com/file/d/1sivGskSPJTXBk-NsNQspVQk9jOdPdY30/view

CONCLUSION

The AI is able to detect ASL letters and phrases distinctly with an accuracy that is satisfactory for real time implementation of hand sign detection.
The software successfully translates the detected hand signs from the user's web cam to standardized English from ASL dataset.
The software first presents an auditory output of the translated letters or phrases.
A text output is then shown as ouput with all the translated letters or phrases depending on the section chosen previously.