ESI Preprints **Not Peer-reviewed**

Empowering Diversity in Education: A Web-Based Tool for Real-Time Sign Language Detection

Shefqet Meda
Canadian Institute of Technology, Albania
International Balkan University, Skopje, North Macedonia
Oresti Leka
Canadian Institute of Technology, Albania
Hiqmet Kamberaj
International Balkan University, North Macedonia

Doi: 10.19044/esipreprint.7.2024.p229

Approved: 11 July 2024
Posted: 13 July 2024

Cite As:
Meda, S., Leka, O., & Kamberaj, H. (2024). *Empowering Diversity in Education: A Web-Based Tool for Real-Time Sign Language Detection.* ESI Preprints.
https://doi.org/10.19044/esipreprint.7.2024.p229

Abstract

This study aims to give an overview of the disabilities faced by people around the world and, in particular, in Albania. It also highlights the primary concern of the deaf community, such as the expression. This study presents a solution to this problem through a real-time sign language detection web application. First, the structural and functional aspects of the application are described by the standard software specification requirements. Then, a description of the software architecture follows. The technological aspect of the application is entirely defined by expressing all the significant implementations of each programming language contained behind the software diagrams and architecture. The main goal is to provide the latest technologies in the market combined with a well-planned and developed image detection application that is scalable, portable, maintainable, and efficient in Empowering Diversity in Education. This technologically up-to-date application is delivered as a full-stack web application. The application produces results that waver between 85%-95% quality off detection; it consists of a bi-partite backend, one for creating and compiling models, namely Python, TensorFlow, and Jupyter Notebooks, and the other part held by Node.JS to power the front-end built with React and

TensorFlow's JS. This study presents a fully developed web-based application that has the potential to impact and improve diversity in education significantly.

Keywords: Diversity in Education, Artificial Intelligence, Machine Learning, Sign Language Detection

Introduction

Freedom of expression, a universally recognised human right, is crucial for the deaf community. The Canadian Museum for Human Rights (1) asserts that language rights are human rights, a principle that underpins our research. We aim to address the communication challenges faced by the deaf community, who often struggle to express themselves due to their speaking impairment, particularly in the context of education.

Millions of people worldwide have severe hearing loss. The overwhelming majority reside in low—and middle-income nations (2), where they frequently lack access to suitable ear and hearing care treatments. Additionally, hundreds of millions of people are at risk for hearing loss from noise exposure, including both industrial and recreational noise, known as *noise pollution* (3). Hearing loss is a severe barrier to those without appropriate solutions.

Public health initiatives can stop many causes of hearing loss. People with hearing loss can realise their full potential through rehabilitation, education, and empowerment.

For example, in Albania, according to the INSTAT (Institute of National Statistics) institution, 6.2% of the entire population suffers from some disability, and 1.5% suffer from speech impairment (4). It is a small number in statistics, but it may be significant for those who live that life. The sign language detection web application intends to tackle the struggles of everyday life for these individuals, serving as a bridge between them and people who might have difficulty understanding them, allowing them the freedom of expression and easing their burden.

The real-time sign language detection web application, combining some of the latest technologies available, intends to create a tool to aid these people in their everyday activities and education. Breaking down the need for specialised devices or even human translators, the so-called *web apps* can turn every smartphone and every computer at the user's disposal into their immediate, instantaneous translator to convey every sign they make into written words. As a result, the receiver of the message is no longer required to know sign language to communicate.

With TensorFlow (5) as its main component for image detection and classification and Python (6) programming language, known for fast

processing speed and machine learning, this web application is a quick and dependable solution that can fit in a pocket. Adding to them a front-end with React.JS (7) upheld by Node.JS (8), both based on JavaScript (9), the most popular programming language at the moment, makes it state-of-the-art software. Especially since Albania had little to no technological involvement in this area, to the best of our knowledge, we hope to shift in a better direction and aim with this technology to empower diversity and inclusion in the education system.

The primary purpose of this study is to give a clear understanding of what a community of individuals is going through as we speak. Secondly, this study provides an option for a possible solution to the problem using current advanced technologies. In particular, we are searching for a solution that implements real-time sign language with TensorFlow and a ReactJS web application.

Furthermore, this study aims to answer the following common questions:

i. What is the current and historical state of the deaf community in Albania?
ii. How does the web application aid and affect that community?

1. Methodology

1.1. Global Analysis

According to the World Health Organization (10), millions of people around the globe live with a disabling hearing impairment, as displayed in **Error! Reference source not found.** in detail.

Grade impairment	Corresponding audiometric ISO value	Performance	Recommendations
No impairment	Less or equal to 25 dB (better ear)	No or very slight hearing problems. Able to hear whispers	
Slight impairment	26-40 dB	Able to hear and repeat words spoken in a normal voice at a distance of one metre.	Counselling because aids may be needed.
Moderate impairment	41-60 dB (better ear)	Can hear and repeat words spoken in a raised voice at a distance of one metre	Hearing aids are usually recommended.
Severe impairment	61-80 dB (better ear)	Can hear some words when shouted into a better ear	Hearing aids are needed. If no hearing aids are available, lip-reading and sign language should be taught.
Profound impairment, including deafness	Greater or equal to 81 dB (better ear)	Can not hear and understand even a shouted voice	Hearing aids may help in understanding words. Additional rehabilitation is needed. Lip-reading and sometimes signing are essential.

Table 1: Grades of hearing impairment (10)

Besides, most of the affected by hearing disabilities can be found in third-world countries. Children who suffer from measles, mumps, rubella, meningitis, and ear infections are more than 30% more likely to have hearing loss. Up to 330 million people worldwide are thought to experience chronic otitis media or ear infections. Chronic ear infections result in hearing loss when untreated and other potentially fatal problems (10). A summary is as follows (10):

- 34 million children have hearing loss disability.
- Nearly one out of every three people over 65 years old are affected by hearing loss disability.
- Noise is a major avoidable cause of hearing loss.
- Sign language and captioning services facilitate communication with deaf and hard-of-hearing people.
- Of those who could benefit from a hearing aid, only 17% use one.
- Globally, 1.5 billion people live with some hearing loss, of which around 430 million require rehabilitation services.

The following shows the importance of that aimed at our study:

- Firstly, we conclude that a real problem needs to be tackled.
- Secondly, we understand the situation, its causes, and whether something can be done.
- Thirdly, the World Health Organization stresses that this is a significant problem in deaf communities in third-world countries. This directly affects Albania (including the countries in the Balkan region), a high third-world-low second-world tier nation. Since the web application will be jump-started in Albania, there is only one way to proceed by knowing the situation of this community in this country to determine if there is a need for this sort of intervention.

As stated above, this nation is developing and struggling. Little to no effort is directed toward marginalised groups of society and their needs. The studies that have been conducted date back to 2015-2016 (11) and no other relevant effort has been made except the following from INSTAT in 2019 (12), displayed in Figure 1, which includes data from 2011 sources (Census 2011).

Figure 1. Limited ability's prevalence according to the impairment type (in percentage) (12).

Figure 1 shows that out of the 6.2% of disabled individuals, 1.5% of the population has a hearing-induced disability, which translates to about 4000 individuals (12). This demonstrates that these situations can be found and reflected in Albania. But why would there be a need for a sign language detection web application for these people to communicate?

The answer lies not with the deaf community but with the rest of the population, as shown in Figure 2 (11). The hearing-impaired community has no trouble expressing itself but is challenged when it wants to live an everyday life and communicate with others. A 2015 survey study conducted in collaboration with the Albanian government, the Finnish Association of the Deaf, and many others found that 68% of people interviewed do not know the Albanian sign language, 25% of them know a little, and only 7% knew it at very well level (11).

Figure 2. Knowledge of Albanian Sign Language among respondents (11).

Thus, most of the population (around 68%) must know about the Albanian sign language. That establishes a group of affected individuals of interest, and there is a gap between them and the rest of the population that we need to try to mend.

During our study of any attempts made to intervene in this situation, it was found that using a Kinect device and an image-capturing device, one can capture images, segment them, and properly label them (13). However, in today's world, it could be more practical because Kinect technology has fallen far behind the likes of PyTorch and, in particular, TensorFlow. Not to mention that these new technologies do not require additional hardware to make real-life detection possible.

We have detected a substratum of individuals who have a disability and need aid. Besides, we have determined the gist of their problem for the rest of the population. Moreover, more effort is required to aid, regress, or solve this issue, with the last attempt being about eight years ago. Therefore, the real-time sign language detection web application could be a groundbreaking intervention and lasting solution for all the parties affected by this situation.

The real-time sign language detection with TensorFlow (5) and ReactJS (7) web applications developed and discussed in this study results from combining the best and most popular technologies. We can use Python (6) with TensorFlow (5) to create a machine learning environment, capture sample images, and turn them into trained detection models easily, quickly, and consistently, making future growth or maintenance easy. TensorFlow's JS enables us to compress all model data into a lightweight and transportable format that can be easily stored and used as a link in the React web

application construction. That computer technology infrastructure may allow every device to become a detection device that everyone can use.

2. Use of Advanced Technologies

Recently, the two fields of Machine Learning (ML) and Artificial Intelligence (AI) have gained much prominence in image detection and construction space (14). Though huge strides have been made, these fields have yet to reach their full potential in efficacity (15). In our approach to tackling an issue that plagues millions worldwide, we decided to implement a study and development methodology based on practicality, speed, user-friendliness, and, as already stated above, efficacity. We can quickly build a three-step path to success using TensorFlow (5), the best that machine learning has yet to offer. The steps consist of building, training, and deploying. Building the basis of the program and a web app allows quick data collection and database implementation. Followed up by training upon gathering ensures a stream of information is fed to the algorithm to maximise the image detection output. Lastly, having the app already running keeps the stream uninterrupted and on a fast track to live deployment. In conjunction with user feedback, all these three key points turn the path into a moldable cycle that can transform the final product to better fit and improve each user experience. With the infrastructure already built, this cycle can be run over and over to remove the language barrier for users, as it can endlessly grow with no drawbacks into a global solution.

Nevertheless, this methodology remains, to date, the best way to break the barrier of words in a real case, a live scenario that will be the basis for bettering numerous lives through a diversity and inclusion process.

Software Requirement Specification

Requirements

Enumerated Functional Requirements

This software works in system-user real-time operation. The user only needs to input visual stimuli for the software to process them independently. Thus, for documentation, we will display the software's functional requirements. Here, Figures 3 – 8 demonstrate the software's basic needs and ideas and the results of its utilisation.

1) The following software functional requirements can be listed (see also Figure 3):Access and activate the camera or video recording device.
2) Detect and label said detection.

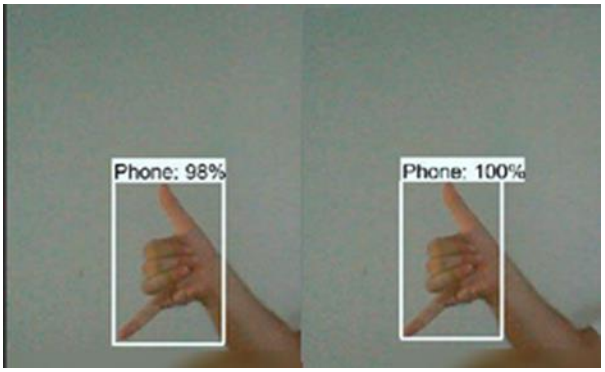


Figure 3. Detect and label detection

Return a detection from one or more elements of the sets shown in Figure 4 and Figure 5.

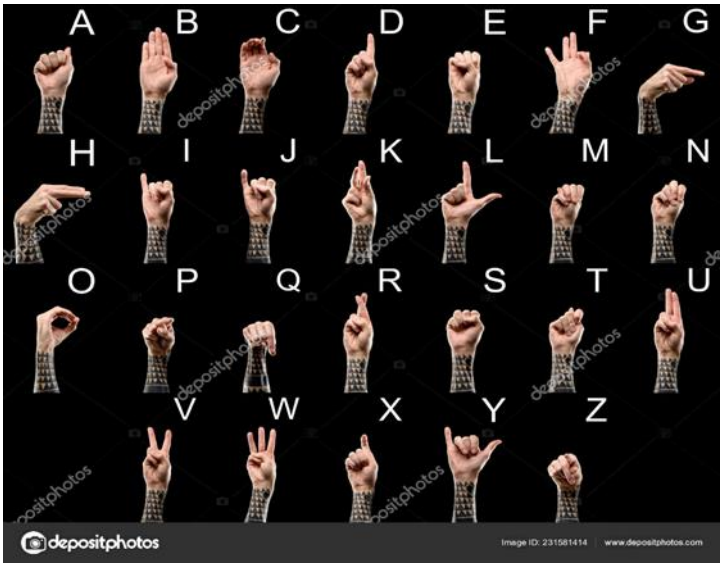


Figure 4. Sign Language. (16)

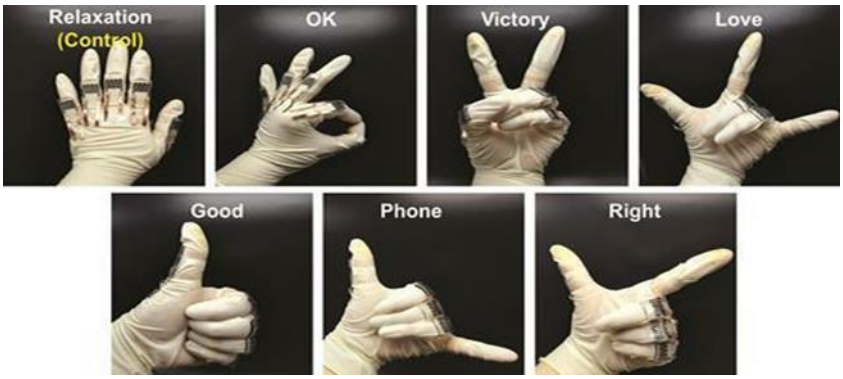
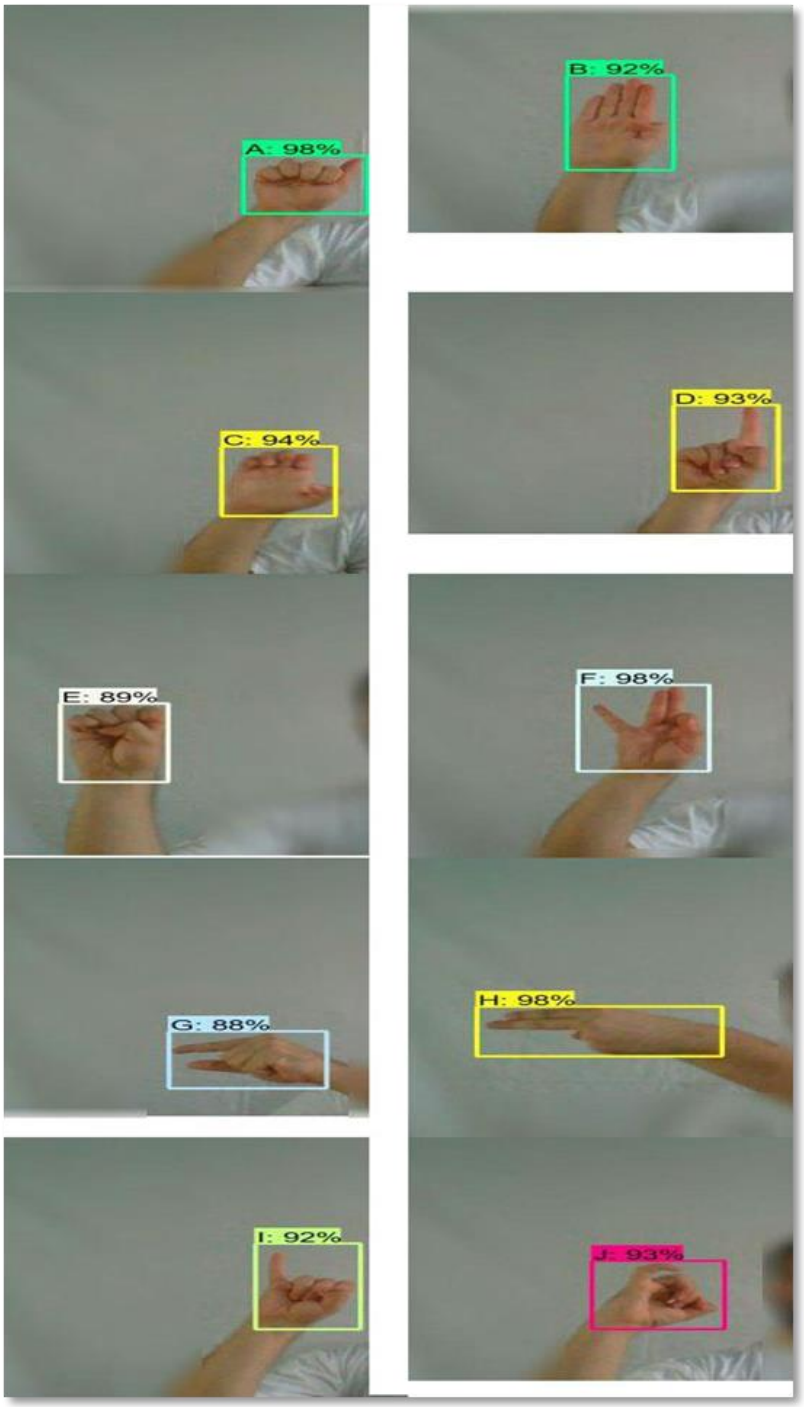


Figure 5. Different Sign Language (17)

Concrete test results, as shown in Figure 6 to Figure 8.

Figure 6: Concrete results (set 1)



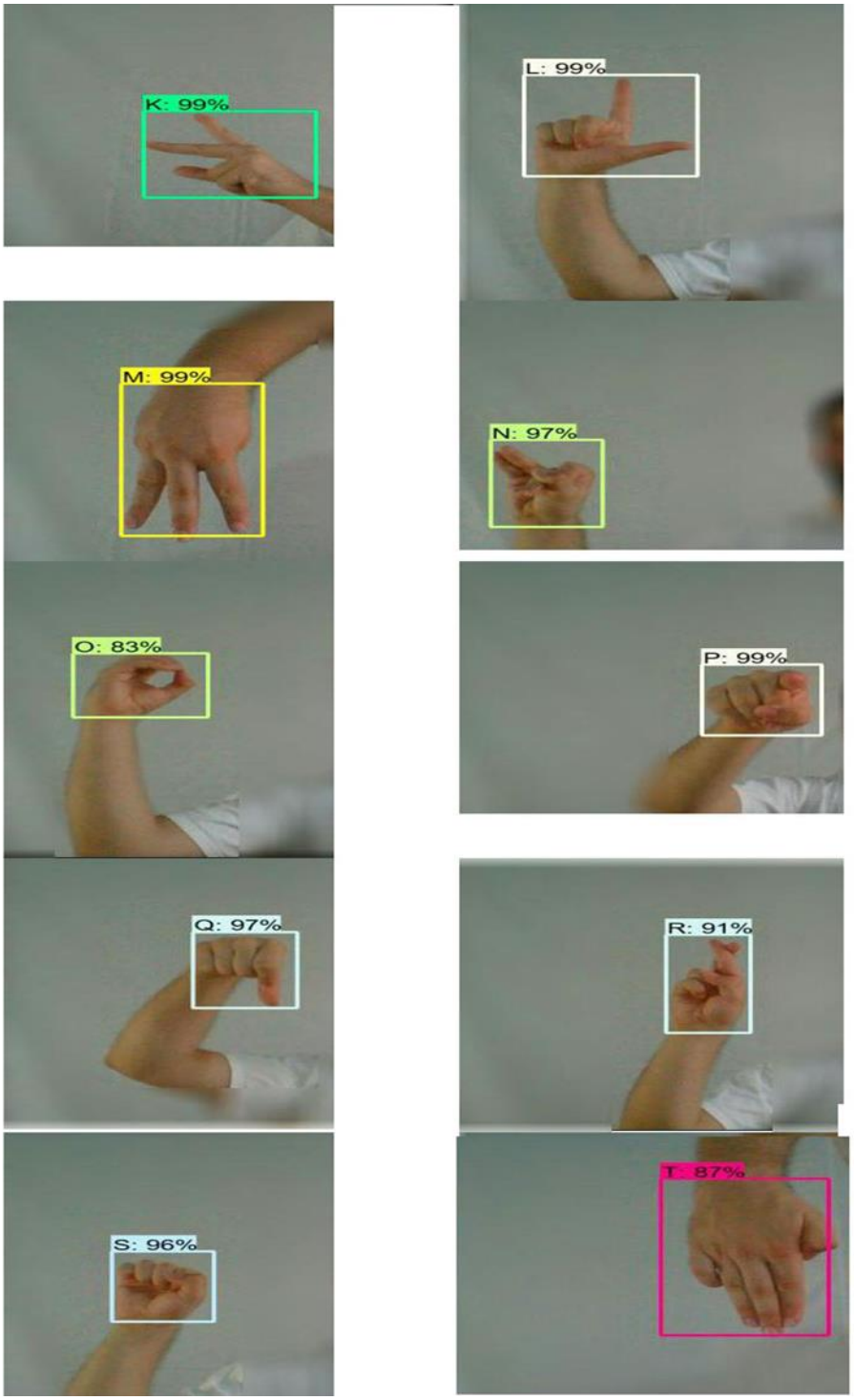


Figure 7: Concrete results (set 2)

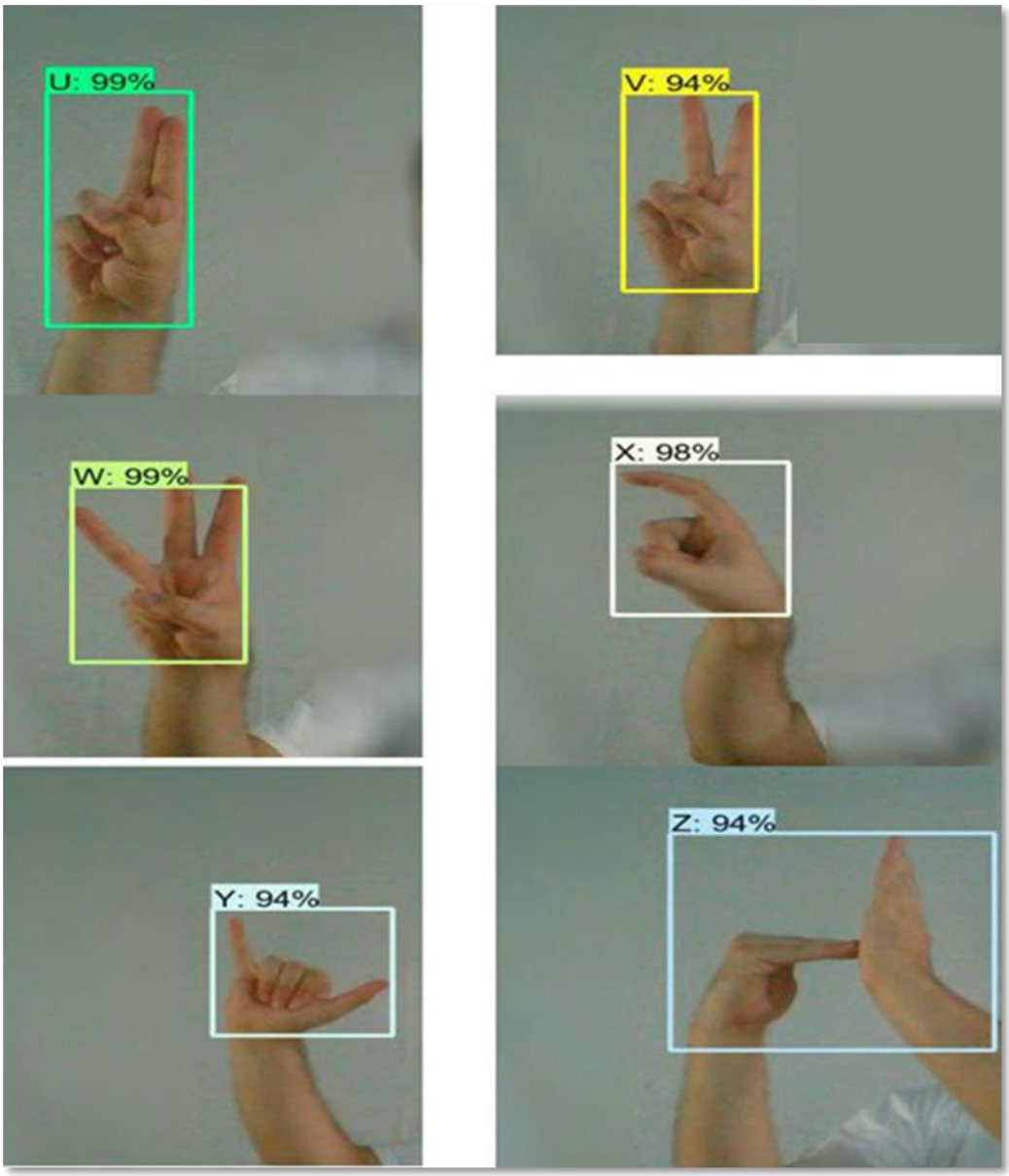


Figure 8: Concrete results (set 3)

Enumerated Non-Functional Requirements

Usability Requirements

This feature concerns the users about how effectively they can learn and use a system, such as:

- Being user-friendly,
- Wide variety of options to choose from,
- Being able to be accessed quickly, and

- Being adaptable for each screen size.

Performance Requirements

- The site should load in at least 3 seconds, according to the standard web page/app loading time.
- Response time when the visual trigger is detected:
- 2 seconds - the limit after which the system reaction does not seem instantaneous;
- 1-2 seconds - when users might have a slow internet connection;
- More than 10 - 12 seconds - when the user has a low-quality capturing device or is not inserting proper triggers.

Security Requirements

- A web application uses HTTPS for security.
- A web application has no data taken from users, thus eliminating the possibility of leaks.

Software Quality Attribute Scenarios

- Usability Scenario

Figure 9 gives a simple visual representation of how a stimulus is introduced in the system.

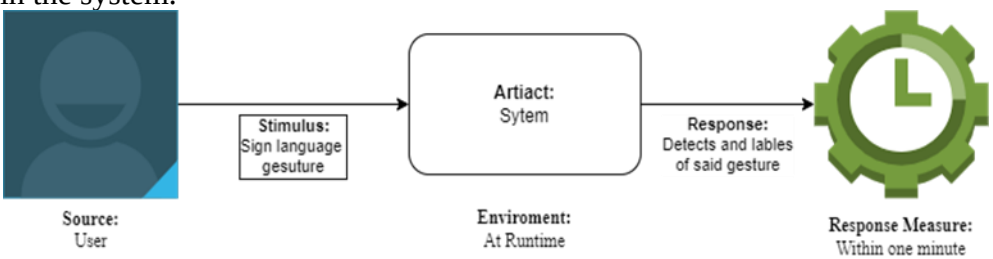


Figure 9. Usability Scenario

Hardware Requirements

This web application does not require additional hardware. Due to TensorFlow (explained in the technology section), users need only a device capable of connecting to the Internet and any image- or video-capturing device.

Software Requirements

We can access the web application from any available web browser. Any Windows system (version eight and above) is preferred for computers and laptop operating systems, and any Linux system released after 2013. As for mobile devices, there should be no issue with usage, but due to the

hardware constraints on the capturing devices, Android 7.0 or a newer version is the best fit.

Software Architecture

The application follows the Model View Controller (MVC) architectural pattern, as sketched in Figure 10. As mentioned in (18), the MVC pattern is made out of three main components:

Model – All business logic is contained in the backend. Here, various algorithms or computations are carried out to provide the front end with the quickest and lightest answer feasible for each request.

View – The front end of an image detection or recognition web application is a visual representation of what the device captures at the moment of usage. Thus, it does not have a fleshed graphical interface like other applications, such as Snapchat, but a live camera feed.

Controller – This is the brain of the application that controls how the data will be displayed. It takes the visual stimuli that get input through the front end/capturing device and sends them to be cross-referenced with the detection models in the backend, then displays the appropriate detection in the front end. Figure 10 shows how MVC works for this application.

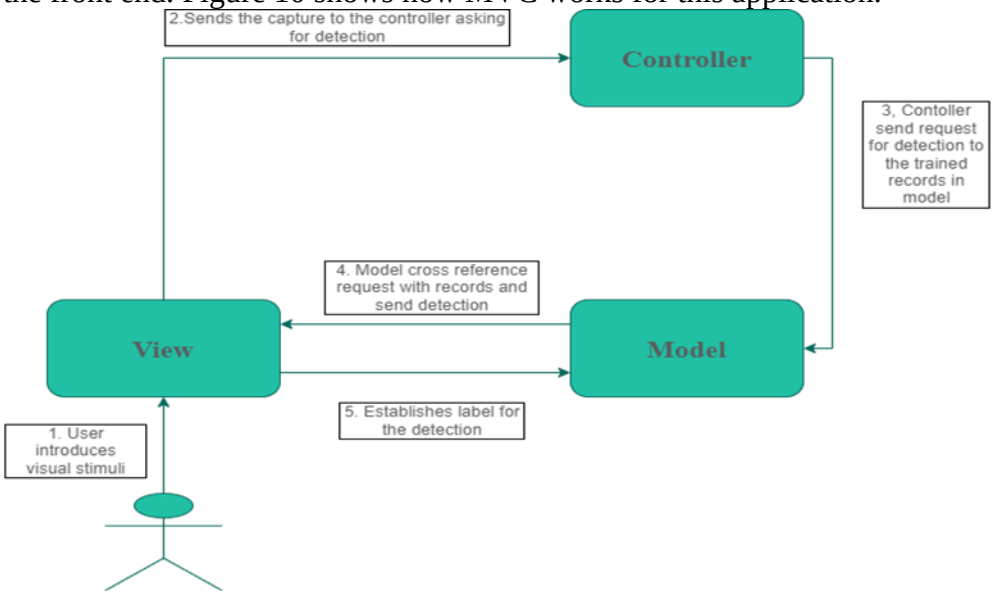


Figure 10. Illustration of how MVC works (18)

Implementation of Object detection

This type of application is a full-stack web application with MVC architecture, used for creating highly effective, scalable, reliable, and efficient applications. The image detection web application consists of two leading technologies, Python and JavaScript, and their sub-packages. The

leading technologies are utilised in constructing this project, such as Python (19), TensorFlow and PyTorch (20), Node.js (8), JavaScript is used (9), 21), ReactJS and TensorFlow's JS (22).

Image Processing and Detection Implementation

After understanding what this web application needs to fulfil in terms of requirements and what technologies have been implemented in developing the software, let's observe the coding factors that build the basis for the upcoming application. We will see the two primary partitions of the back end of this application, namely the *image collection* and *detection training*.

Image Collection

To create the application's basis, we first needed to have Python access to the capturing device's camera, which, in this case, was our laptop. Jupyter Notebook comes in handy in this situation. Because a compiler can be used previously and might have libraries and other packages that may overlap or intrude on our work, creating a separate virtual environment in Jupyter allows us to have only the libraries needed to develop this software. Figures 11 to 22 show the code snippets displaying how this software works. Figure 11 is a screenshot of the imported external packages.

```
!pip install opencv-python
```

```
# Import opencv
import cv2

# Import uuid
import uuid

# Import Operating System
import os

# Import time
import time
```

Figure 11. Import Dependencies

OpenCV-Python stands for "open computer vision," a library that, by installing it, will allow Python to access the device's video camera, allowing it to capture images. Then, we get some imports of the basic needs for the application:
import cv2: imports open cv so it can be used.
import uuid: imports Universally Unique Identifier so we can name every image uniquely, which is very important.
import os: imports operating system for the notebooks to work with

import time: imports the concept of time into the code to time the image capture sequence.

After setting up the basic dependencies, we can start the image collection process. First, there is a need to create labels and set the number of images that will be collected per label, like:

labels = [#insertLabels]

number_imgs = 10 (preferably)

Note that this states in the code what is needed; we then proceed to create these files with appropriate paths:

IMAGES_PATH = os. path.join('TensorFlow,' 'workspace,' 'images,' 'collected images')

Since we are working in a Jupyter Notebook, we added a verification script to determine the path depending on the operating system used, as shown in Figure 12.

```
if not os.path.exists(IMAGES_PATH):
    if os.name == 'posix':
        !mkdir -p {IMAGES_PATH}
    if os.name == 'nt':
         !mkdir {IMAGES_PATH}
for label in labels:
    path = os.path.join(IMAGES_PATH, label)
    if not os.path.exists(path):
        !mkdir {path}
```

Figure 12. OS Verification

The proper paths will be created for Windows or Linux, depending on the OS used. As illustrated in Figure 13, we can proceed with image collection with the dependencies imported and the label paths created.

```
for label in labels:
    cap = cv2.VideoCapture(0)
    print('Collecting images for {}'.format(label))
    time.sleep(5)
    for imgnum in range(number_imgs):
        print('Collecting image {}'.format(imgnum))
        ret, frame = cap.read()
        imgname = os.path.join(IMAGES_PATH,label,label+'.'+'{}.jpg'.format(str(uuid.uuid1())))
        cv2.imwrite(imgname, frame)
        cv2.imshow('frame', frame)
        time.sleep(3)

        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
cap.release()
cv2.destroyAllWindows()
```

Figure 13. Image Capture

It works by first searching all the labels created, thus gauging the number of files that need images collected. Then, it enables OpenCV and gains access to the camera or capture device. It collects images for each label, gives them unique identifiers, and terminates the function after fulfilling its process.

Once the images are collected, we power up Anaconda, a library that allows us to label the elements in the pictures we are interested in detecting.

We first install PyQT5 (23), a library that will enable the input of labels on the images.

!pip install --upgrade pyqt5 lxml

Once installed, we set up the files and the file path where these labels will be stored:

LABELIMG_PATH = os. path.join('TensorFlow', 'labelling')

In our case, we did not have the proper file format to proceed. Hence, the git hub repository of *tzutalin* (labeling 2022) came in handy to solve the problem and build the paths, so we turned it into a script if there was one. As shown in Figure 14, future maintenance will be needed.

```
if not os.path.exists(LABELIMG_PATH):
    !mkdir {LABELIMG_PATH}
    !git clone https://github.com/tzutalin/labelImg {LABELIMG_PATH}
```

Figure 14. Labeling path

Since paths were built, an OS verification script must be run, as depicted in Figure 15.

```
if os.name == 'posix':
    !cd {LABELIMG_PATH} && make qt5py3
if os.name =='nt':
    !cd {LABELIMG_PATH} && pyrcc5 -o libs/resources.py resources.qrc
```

Figure 15. OS Verification Script

In this case, the verification script also integrates the PyQT5 library into the path, allowing to run the command to activate the label program:

cd {LABELIMG_PATH} && python labelImg.py

Having the program up and running, we could label every entry for each image, as shown in Figure 16.

Figure 16: Labeling detections

The labelling library allows us to select the desired element for detection for each image. This is done manually and is time-consuming, but we shift the images into a test and training folder once it is done. That successfully concludes the image collection partition of this software development process.

Detection Training

This partition of the code took place in another Jupyter Notebook. Thus, some basic dependencies need to be set in place again. This part of the notebook focuses on training object detection and creating trained models to be tested and implemented in the web application as pre-trained models. It all begins by importing the OS into the notebook, as seen in Figure 17.

```
CUSTOM_MODEL_NAME = 'my_ssd_mobnet'
PRETRAINED_MODEL_NAME = 'ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8'
PRETRAINED_MODEL_URL = 'http://download.tensorflow.org/models/object_detection/tf2/20200711/ssd_mobilenet_v2_fpnlite_320x320_coc
TF_RECORD_SCRIPT_NAME = 'generate_tfrecord.py'
LABEL_MAP_NAME = 'label_map.pbtxt'
```

Figure 17. Directories

CUSTOM_MODEL_NAME: Defines the name the trained detection model will have.
PRETRAINED_MODEL_NAME: TensorFlow's default trained detection model can later be changed to suit the software's needs.
PRETRAINED_MODEL_URL: The site where the pre-trained models are retrieved.
TF_RECORD_SCRIPT_NAME: Generating record files where the training and testing will be done.
LABEL_MAP_NAME: Creation of a label map.
After setting the pre-trained model variables, the software must create all the paths it will use for training, as in Figure 18.

```
paths = {
    'WORKSPACE_PATH': os.path.join('Tensorflow', 'workspace'),
    'SCRIPTS_PATH': os.path.join('Tensorflow','scripts'),
    'APIMODEL_PATH': os.path.join('Tensorflow','models'),
    'ANNOTATION_PATH': os.path.join('Tensorflow', 'workspace','annotations'),
    'IMAGE_PATH': os.path.join('Tensorflow', 'workspace','images'),
    'MODEL_PATH': os.path.join('Tensorflow', 'workspace','models'),
    'PRETRAINED_MODEL_PATH': os.path.join('Tensorflow', 'workspace','pre-trained-models'),
    'CHECKPOINT_PATH': os.path.join('Tensorflow', 'workspace','models',CUSTOM_MODEL_NAME),
    'OUTPUT_PATH': os.path.join('Tensorflow', 'workspace','models',CUSTOM_MODEL_NAME, 'export'),
    'TFJS_PATH':os.path.join('Tensorflow', 'workspace','models',CUSTOM_MODEL_NAME, 'tfjsexport'),
    'TFLITE_PATH':os.path.join('Tensorflow', 'workspace','models',CUSTOM_MODEL_NAME, 'tfliteexport'),
    'PROTOC_PATH':os.path.join('Tensorflow','protoc')
}
```

Figure 18. Paths scripts

The script builds every path needed for the resources that a training model takes, as listed in Figure 19.

```
paths
```

```
{'WORKSPACE_PATH': 'Tensorflow\\workspace',
 'SCRIPTS_PATH': 'Tensorflow\\scripts',
 'APIMODEL_PATH': 'Tensorflow\\models',
 'ANNOTATION_PATH': 'Tensorflow\\workspace\\annotations',
 'IMAGE_PATH': 'Tensorflow\\workspace\\images',
 'MODEL_PATH': 'Tensorflow\\workspace\\models',
 'PRETRAINED_MODEL_PATH': 'Tensorflow\\workspace\\pre-trained-models',
 'CHECKPOINT_PATH': 'Tensorflow\\workspace\\models\\my_ssd_mobnet',
 'OUTPUT_PATH': 'Tensorflow\\workspace\\models\\my_ssd_mobnet\\export',
 'TFJS_PATH': 'Tensorflow\\workspace\\models\\my_ssd_mobnet\\tfjsexport',
 'TFLITE_PATH': 'Tensorflow\\workspace\\models\\my_ssd_mobnet\\tfliteexport',
 'PROTOC_PATH': 'Tensorflow\\protoc'}
```

Figure 19. Paths

Then, all that remains is an OS verification script to build these files again in the proper order for the appropriate system, as shown in Figure 20.

```
for path in paths.values():
    if not os.path.exists(path):
        if os.name == 'posix':
            !mkdir -p {path}
        if os.name == 'nt':
            !mkdir {path}
```

Figure 20. OS verification scripts

With all of these components in place, the next step is to incorporate TensorFlow and TensorFlow object detection into our code and software; this will serve as the foundation for training the previously collected array of images; this is done as in Figure 21.

```
# Install Tensorflow Object Detection
if os.name=='posix':
    !apt-get install protobuf-compiler
    !cd Tensorflow/models/research && protoc object_detection/protos/*.proto --python_out=. && cp object_detection/packages/tf2/

if os.name=='nt':
    url="https://github.com/protocolbuffers/protobuf/releases/download/v3.15.6/protoc-3.15.6-win64.zip"
    wget.download(url)
    !move protoc-3.15.6-win64.zip {paths['PROTOC_PATH']}
    !cd {paths['PROTOC_PATH']} && tar -xf protoc-3.15.6-win64.zip
    os.environ['PATH'] += os.pathsep + os.path.abspath(os.path.join(paths['PROTOC_PATH'], 'bin'))
    !cd Tensorflow/models/research && protoc object_detection/protos/*.proto --python_out=. && copy object_detection\\packages\\
    !cd Tensorflow/models/research/slim && pip install -e .
```

Figure 21. TensorFlow object detection

This brings the software to a stage where:

- The images have been stored and labelled.
- All the paths needed are created.
- TensorFlow object detection has been installed.

It is time to start the training process, which follows these steps.
Step 1: Creation of a label map for all the labels created until now, as in Figure 22.

```
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},
{'name':'Z', 'id':26},
{'name':'Good', 'id':27},
{'name':'Love', 'id':28},
{'name':'Relax', 'id':29},
{'name':'ThankYou', 'id':30},
{'name':'Ok', 'id':31}
]

with open(files['LABELMAP'], 'w') as f:
    for label in labels:
        f.write('item { \n')
        f.write('\tname:\'{}\'\n'.format(label['name']))
        f.write('\tid:{}\n'.format(label['id']))
        f.write('}\n')
```

Figure 22. Label map

Step 2: Create the train and test record files. These will go on to be trained:
python {files['TF_RECORD_SCRIPT']} -x {os.path.join(paths['IMAGE_PATH'], 'train')} -l {files['LABELMAP']} -o {os.path.join(paths['ANNOTATION_PATH'], 'train. record')}
python {files['TF_RECORD_SCRIPT']} -x {os.path.join(paths['IMAGE_PATH'], 'test')} -l {files['LABELMAP']} -o {os.path.join(paths['ANNOTATION_PATH'], 'test. record')}
Step 3: With the labels and record files in place, the program is ready to begin training with the command:

```
pythonTensorflow\models\research\object_detection\model_main_tf2.py--model_dir=TensorFlow\workspace\models\my_ssd_mobnet
--pipeline_config_path=Tensorflow\workspace\models\my_ssd_mobnet\pipeline.config--num_train_steps=10000
```

This command starts the training procedure. Every image taken is cross-referenced with each other for 10,000 steps, which takes a significant amount of time and processing power. (Note: A weak or old device might break down, so a powerful device is suggested.) The detection model is complete at the end of the process, and development can move forward.

Evaluation and Testing

Evaluation

After training is finished, we can do statistical analysis using the following Python command line:

PythonTensorflow\models\research\object_detection\model_main_tf2.py--model_dir=TensorFlow\workspace\models\my_ssd_mobnet--pipeline_config_path=Tensorflow\workspace\models\my_ssd_mobnet\pipeline.config--checkpoint_dir=TensorFlow\workspace\models\my_ssd_mobnet

Output is shown in Figure 23.

```
Accumulating evaluation results...
DONE (t=0.53s).
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.870
 Average Precision  (AP) @[ IoU=0.50       | area=   all | maxDets=100 ] = 1.000
 Average Precision  (AP) @[ IoU=0.75       | area=   all | maxDets=100 ] = 1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.825
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.875
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.877
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.877
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.877
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.833
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.881
INFO:tensorflow:Eval metrics at step 10000
```

Figure 23. Evaluations results

The ideal result would be to get as close as one, and the evaluation results show that the average precision is 0.875 and the average recall is 0.881—auspicious results.

Thanks to the Tensor Board subpackage of TensorFlow, further evaluation is possible. The results are shown in Figure 24.

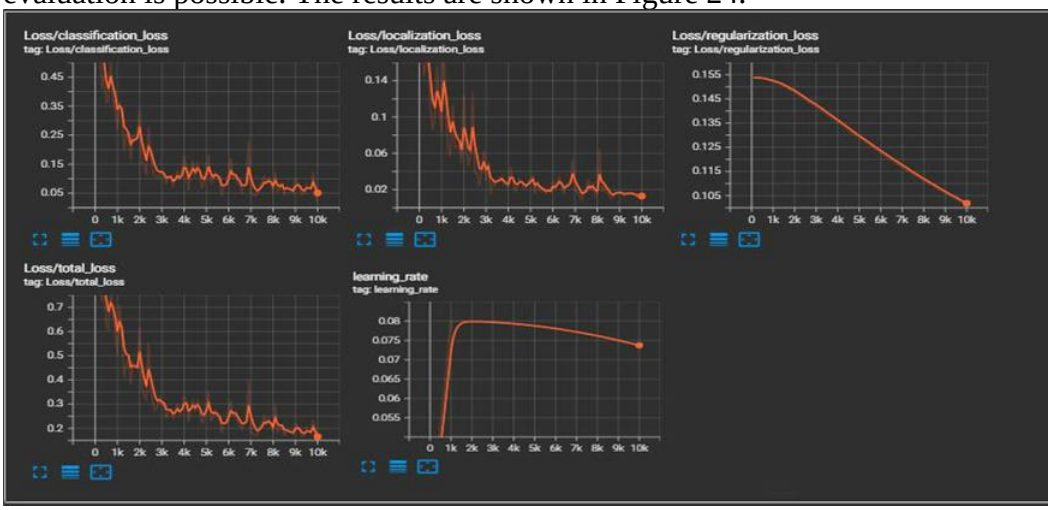


Figure 24. Evaluation graph

Interestingly, every loss metric for this training process shows a decreasing trend, and the learning rate shows primarily an increasing trend. With these evaluations, we can test the models.

Testing

To test if the detection works, we need to run a code snippet to check if any detection is happening in the test partition. The procedure is displayed in Figure 25.

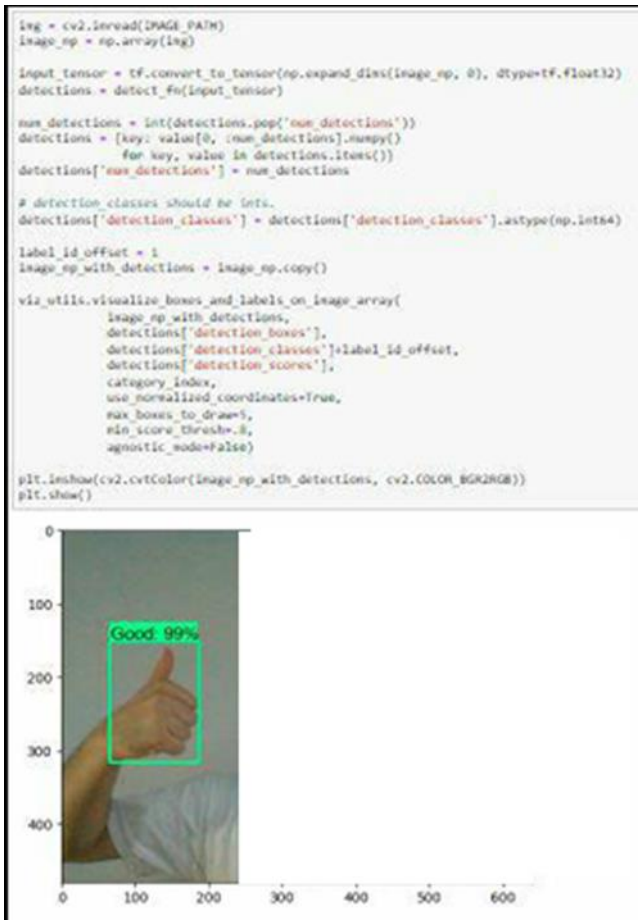


Figure 25: Test result

Notably, the script runs successfully, so the image collection and training have been conducted successfully. All that remains now is making these partitions in TensorFlow's JS format and sending them to the React web application. We implemented the Web-based Tool for Real-time Sign Language Detection by combining TensorFlow and ReactJS.

Additionally, we have provided Jupyter notebooks for each experiment and other necessary tools to facilitate further exploration and analysis. By making our code openly accessible, we hope to encourage collaboration and contribute to the reproducibility of research in the field.

Conclusions

The primary goal of this study was to show the current state of the global dead community and its data (10). It aimed to demonstrate the same from a more regional standpoint, using Albanian data and perspectives (4). Besides, it helps to bring forward the issue we were trying to tackle, the fringe between people and those with this disability. The bridge presented in Section 1 should be crossed with the implementation of the web application and the utilisation of the technologies discussed in this study. When it comes directly down to real-time sign language detection with TensorFlow and a ReactJS web application, it demonstrates the following qualities:

i. One-tap activation means there is little to no difficulty in usage;
ii. almost instantaneous response between visualisation and classification,
iii. a dynamic, ever-growing library of detections, and
iv. adaptability to every device.

These contribute to making an application that would otherwise require sophisticated hardware and specialised individuals into a straightforward, usable, and portable web app. The current application supports the basic English alphabet and some hand signs, making for about 31 possible detections and compelling the usage of these technologies. Because of Python and TensorFlow, the application is easily moldable and growable, making the potential use limitless and timeless. This first version is a small but secure step in the right direction to better the lives of a community.

Software Availability

The source code for our project is available on GitHub (24) at https://github.com/Oresti9919/SignLanguageApp/tree/main.

The repository contains the following three elements:

1. Application: Created on July 22, 2023.
2. Image Gathering: Created on July 22, 2023.
3. Training and Preview: Created on July 22, 2023.

Authors Contributions

Shefqet Meda: He did conceptualisation, provided the resources, and supervised the software writing, development, and testing. Oresti Leka: He wrote, tested, and evaluated data curation, prepared the figures, and analysed the software results. He also wrote the first draft of the paper. Hiqmet Kamberaj: He was involved in methodology, validation, investigation,

formal analysis, funding acquisition, and preparing the final version of the edited paper, including the English language check.

Acknowledgments: The author (H. K.) thanks the International Balkan University's support.

Conflict of Interest: The authors reported no conflict of interest.

Data Availability: All data are included in the content of the paper.

Funding Statement: The authors did not obtain any funding for this research.

References:

1. Canadian Museum for Human Rights. (2019, September). *Language rights are human rights*. Retrieved from https://humanrights.ca/story/language-rights-are-human-rights
2. World Health Organization. (2024, February 2). *Deafness and hearing loss*. Retrieved from https://www.who.int/news-room/fact-sheets/detail/deafness-and-hearing-loss
3. GBD 2019 Hearing Loss Collaborators. (2021, March 13). Hearing loss prevalence and years lived with disability, 1990–2019: Findings from the Global Burden of Disease Study 2019. Retrieved from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7960691/#BIB12
4. INSTAT. (2019). *Profili i personave me aftësi të kufizuar në Shqipëri*. Retrieved from https://www.instat.gov.al/media/3706/profili_i__personave_me_aft_si_t__kufizuar_n__shqip_ri.pdf
5. Géron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*. O'Reilly.
6. Ramalho, L. (2022). *Fluent Python*. O'Reilly Media.
7. Banks, A., & Porcello, E. (2020). *Learning React: Modern Patterns for Developing React Apps*. O'Reilly Media.
8. Casciaro, M., & Mammino, L. (2020). *Node.js Design Patterns - Third Edition: Design and Implement Production-Grade Node.js Applications Using Proven Patterns and Techniques*. Packt Publishing.
9. Osmani, A. (2023). *Learning JavaScript Design Patterns: A JavaScript and React Developer's Guide*. O'Reilly Media.
10. World Health Organization. (2024, April 23). *Deafness*. Retrieved from https://www.who.int/news-room/facts-in-pictures/detail/deafness

11. Finnish Association of the Deaf. (2016). *Deaf people in Albania in 2015: A survey study* (I. Lahtinen & P. Rainò, Eds.). Gent Grafik.
12. INSTAT. (2019). Retrieved from http://www.instat.gov.al/media/3706
13. Eriglen, G., Alda, K., & Bruno, G. (2015). A real-time vision-based system for recognition of static dactyls of Albanian alphabet. In Finnish Association of the Deaf, *Deaf people in Albania*.
14. Rabie, A. R., et al. (2022). Convolution neural network-based automatic localization of landmarks on lateral X-ray images. *Multimedia Tools and Applications*, 81, 37403–37415.
15. Gates, B. (2023, December 26). *AI is about to supercharge the innovation pipeline*. LinkedIn.
16. Depositphotos.com. *Set of deaf and dumb language*. Retrieved from https://depositphotos.com/photo/set-deaf-dumb-language-tattooed-male-hands-latin-alphabet-isolated-231581414.html
17. Royal Society of Chemistry. (2016, August 23). *3D graphene adds dimension to deaf–mute communication*. Retrieved from https://www.chemistryworld.com/news/3d-graphene-adds-dimension-to-deaf-mute-communication/1017315.article
18. Mozilla.org. (2024). *MVC*. Retrieved from https://developer.mozilla.org/en-US/docs/Glossary
19. RedMonk. (2024, March 8). *The RedMonk programming language rankings: January 2024*. Retrieved from https://redmonk.com/sogrady/2024/03/08/language-rankings-1-24/
20. Google Trends. (2024, January 28). *PyTorch-TensorFlow*. Retrieved from https://trends.google.com/trends/explore?geo=US&q=PyTorch,TensorFlow
21. Yankiver, D. (2021, May 31). *5 major companies that use Node.js and why*. Medium. Retrieved from https://medium.com/nerd-for-tech/5-major-companies-that-use-node-js-and-why-211fb5cc267d
22. Built With. (2024, February). *React usage statistics*. Retrieved from https://trends.builtwith.com/javascript/react
23. Fitzpatrick, M. (2022). *Create GUI Applications with Python & Qt5: The hands-on guide to making apps with Python*. Kindle Edition.
24. Leka, O., Meda, S., & Kamberaj, H. (2023, July). *SignLanguageApp*. Retrieved from https://github.com/Oresti9919/SignLanguageApp/tree/main