

A  
Major Project  
On  
**ML BASED REAL TIME SIGN LANGUAGE DETECTION**  
(Submitted in partial fulfillment of the requirements for the award of Degree)  
BACHELOR OF TECHNOLOGY

in  
COMPUTER SCIENCE AND ENGINEERING

By  
P. Rishi Sanmitra (177R1A0548)  
K. Lalithanjana (177R1A0525)  
V. V. Sai Sowmya (177R1A0553)

Under the Guidance of  
**Dr. T. S. Mastan Rao**  
(Associate Professor)



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**CMR TECHNICAL CAMPUS**

**UGC AUTONOMOUS**

(Accredited by NAAC, NBA, Permanently Affiliated to JNTUH, Approved by AICTE, New Delhi)

Recognized Under Section 2(f) & 12(B) of the UGC Act.1956,

Kandlakoya (V), Medchal Road, Hyderabad-501401.

**2017-21**

# DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING



## CERTIFICATE

This is to certify that the project entitled “**ML BASED REAL TIME SIGN LANGUAGE DETECTION**” being submitted by **P. RISHI SANMITRA (177R1A0548), K. LALITHANJANA (177R1A0525) & V. V. SAI SOWMYA (177R1A0553)** in partial fulfillment of the requirements for the award of the degree of B.Tech in Computer Science and Engineering to the Jawaharlal Nehru Technological University, Hyderabad, is a record of bonafide work carried out by him/her under our guidance and supervision during the year 2020-21.

The results embodied in this thesis have not been submitted to any other University or Institute for the award of any degree or diploma.

**Dr. T. S. Mastan Rao**  
**Associate Professor**  
**INTERNAL GUIDE**

**Dr. A. Raji Reddy**  
**DIRECTOR**

**Dr. K. Srujan Raju**  
**HoD**

**EXTERNAL EXAMINER**

**Submitted for viva voce Examination held on \_\_\_\_\_**



## ACKNOWLEDGEMENT

Apart from the efforts of us, the success of any project depends largely on the encouragement and guidelines of many others. We take this opportunity to express our gratitude to the people who have been instrumental in the successful completion of this project. We take this opportunity to express my profound gratitude and deep regard to my guide.

**Dr. T. S. Mastan Rao**, Associate Professor for his exemplary guidance, monitoring and constant encouragement throughout the project work. The blessing, help and guidance given by him shall carry us a long way in the journey of life on which we are about to embark.

We also take this opportunity to express a deep sense of gratitude to Project Review Committee (PRC) Coordinators: **Mr. B. P. Deepak Kumar, Mr. J. Narasimha Rao, Mr. K. Murali, Dr. Suwarna Gothane and Mr. B. Ramji** for their cordial support, valuable information and guidance, which helped us in completing this task through various stages.

We are also thankful to **Dr. K. Srujan Raju**, Head, Department of Computer Science and Engineering for providing encouragement and support for completing this project successfully.

We are obliged to **Dr. A. Raji Reddy**, Director for being cooperative throughout the course of this project. We would like to express our sincere gratitude to Sri. **Ch. Gopal Reddy**, Chairman for providing excellent infrastructure and a nice atmosphere throughout the course of this project.

The guidance and support received from all the members of **CMR Technical Campus** who contributed to the completion of the project. We are grateful for their constant support and help.

Finally, we would like to take this opportunity to thank our family for their constant encouragement, without which this assignment would not be completed. We sincerely acknowledge and thank all those who gave support directly and indirectly in the completion of this project.

**P. RISHI SANMITRA (177R1A0548)**

**K. LALITHANJANA (177R1A0525)**

**V. V. SAI SOWMYA (177R1A0553)**

## **ABSTRACT**

The main purpose of this project is to design a system for the differently abled people to communicate with others with ease. The objective is to design a Machine Learning model that learns the different gestures made by differently abled people to converse using Sign Language. The approach to the problem is to take a set of pictures of the gesture made and label these pictures with the appropriate meaning of it. This approach allows the model to function in multiple ways, because there are different standards of sign language in different geographical locations, for example we have the American Standards of Sign Language and these standards are different from that of the Indian Standards of Sign Language. This project allows us to train the model as per the requirement, so we can train it on the American Standards as well as the Indian Standards. The working principle is as follows; Initially we write some code to automate the picture taking process, once the pictures have been taken we use the LabelImg software to segregate these images into the appropriate labels. These labels are named in such a way that they express the meaning of the gesture made. Once the labelling of the images has been done we have two sets of files for each image taken, one which has the actual image in it and the other being an XML file which contains information of where the model should be looking in the image during the training process. Once these files are generated, the training process begins, where the Machine is going to use a Deep Learning SSD ML algorithm to extract features from the desired image. Finally after the model has been trained, it allows for the Sign Language Detection part to begin. To achieve the detection we are using the TensorFlow Object Detection module, here the extracted features from the images taken are passed onto the TensorFlow module which is going to make comparisons with the real time video present in the frame. On detection of any of these features it is going to generate a bounding box around the gesture and make the prediction. The prediction is going to be the same as the label of the image, hence it is very important to understand the gesture made so as to name the label correctly, a wrongly named label could result in a wrong prediction made.

## **LIST OF FIGURES**

<b>FIGURE NO</b>	<b>FIGURE NAME</b>	<b>PAGE NO</b>
Figure 4.1	Project Architecture	9
Figure 4.2	Use Case Diagram	11
Figure 4.3	Class Diagram	12
Figure 4.4	Sequence Diagram	13
Figure 4.5	Activity Diagram	14

## LIST OF SCREENSHOTS

<b>SCREENSHOT NO</b>	<b>SCREENSHOT NAME</b>	<b>PAGE NO</b>
Screenshot 6.1	LabelImg Software	24
Screenshot 6.2	Images for Training in Grayscale	25
Screenshot 6.3	Loss of Machine Learning Model	26
Screenshot 6.4	Loss at each Iteration	27
Screenshot 6.5	Evaluation Results and Evaluation Metrics	28
Screenshot 6.6	Gesture Recognition for No	29
Screenshot 6.7	Gesture Recognition for ILoveYou and No	30

# TABLE OF CONTENTS

<b>ABSTRACT</b>	i
<b>LIST OF FIGURES</b>	ii
<b>LIST OF SCREENSHOTS</b>	iii
<b>1. INTRODUCTION</b>	<b>1</b>
1.1 PROJECT SCOPE	1
1.2 PROJECT PURPOSE	1
1.3 PROJECT FEATURES	1
<b>2. LITERATURE SURVEY</b>	<b>2</b>
<b>3. SYSTEM ANALYSIS</b>	<b>4</b>
3.1 PROBLEM DEFINITION	4
3.2 EXISTING SYSTEM	4
3.2.1 LIMITATIONS OF EXISTING SYSTEM	5
3.3 PROPOSED SYSTEM	5
3.3.1 ADVANTAGES OF PROPOSED SYSTEM	5
3.4 FEASIBILITY STUDY	6
3.4.1 ECONOMIC FEASIBILITY	6
3.4.2 TECHNICAL FEASIBILITY	7
3.4.3 SOCIAL FEASIBILITY	7
3.5 HARDWARE & SOFTWARE REQUIREMENTS	7
3.5.1 HARDWARE REQUIREMENTS	7
3.5.2 SOFTWARE REQUIREMENTS	8
<b>4. ARCHITECTURE</b>	<b>9</b>
4.1 PROJECT ARCHITECTURE	9
4.2 DESCRIPTION	9



4.3	USE CASE DIAGRAM	11
4.4	CLASS DIAGRAM	12
4.5	SEQUENCE DIAGRAM	13
4.6	ACTIVITY DIAGRAM	14
<b>5.</b>	<b>IMPLEMENTATION</b>	<b>15</b>
5.1	SAMPLE CODE	15
<b>6.</b>	<b>SCREENSHOTS</b>	<b>24</b>
<b>7.</b>	<b>TESTING</b>	<b>31</b>
7.1	INTRODUCTION TO TESTING	31
7.2	TYPES OF TESTING	31
7.2.1	UNIT TESTING	31
7.2.2	INTEGRATION TESTING	31
7.2.3	FUNCTIONAL TESTING	32
7.3	TEST CASES	32
7.3.1	OBJECT DETECTION	32
<b>8.</b>	<b>CONCLUSION &amp; FUTURE SCOPE</b>	<b>33</b>
8.1	PROJECT CONCLUSION	33
8.2	FUTURE SCOPES	33
<b>9.</b>	<b>BIBLIOGRAPHY</b>	<b>34</b>
9.1	REFERENCES	34
9.2	WEBSITES	35



# **1. INTRODUCTION**

# 1. INTRODUCTION

## 1.1 PROJECT SCOPE

The project title as “ML Based Real Time Sign Language Detection” is an application using which a differently abled person can communicate with the people around them with ease. Using this application, the user need not hire an interpreter to help them communicate with others. Using this application, the opposite person can record what the user is trying to convey and get the translation for the gestures made in text format. Using this application the user is able to communicate better with others in their day to day lives.

## 1.2 PROJECT PURPOSE

The project is designed to facilitate a better lifestyle to the differently abled people. Through this project we aim at providing a platform for the differently abled people to communicate with others irrespective of whether they know sign language or not. We aim at diminishing the boundaries created by various medical conditions that restrict the differently abled people to communicate with others. This not only makes them independent, but also provides them with a better lifestyle.

## 1.3 PROJECT FEATURES

The main core feature of this project is that it detects the sign as a complete word instead of a series of alphabets. The other core feature of this project is giving labels to the gesture made. Since we give labels to the gestures made, it can be labelled using any language, giving us an opportunity to include the multi-language feature.

## **2. LITERATURE SURVEY**

## 2. LITERATURE SURVEY

The first approach in relation to sign language recognition was by Bergh in 2011 [2]. Haar wavelets and database searching were employed to build a hand gesture recognition system. Although this system gives good results, it only considers six classes of gestures. Many types of research have been carried out on different sign languages from different countries. For example, a BSL recognition model, which understands finger-spelled signs from a video, was built [3]. As Initial, a histogram of gradients (HOG) was used to recognize letters, and then, the system used hidden Markov models (HMM) to recognize words. In another paper, a system was built to recognize sentences made of 3-5 words. Each word ought to be one of 19 signs in their thesaurus. Hidden Markov models have also been used on extracted features [4].

In 2011, a real time American Sign Language recognition model was proposed utilizing Gabor filter and random forest [5]. A dataset of colour and depth images for 24 different alphabets was created. An accuracy of 75% was achieved utilizing both colour and complexity images, and 69% using depth images only. Depth images were only used due to changes in the illumination and differences in the skin pigment. In 2013, a multilayered random forest was also used to build a real time ASL model [6]. The system recognizes signs through applying random forest classifiers to the combined angle vector. An accuracy of 90% was achieved by testing one of the training images, and an accuracy of 70% was achieved for a new image. An American Sign Language alphabet recognition system was first built by localizing hand joint gestures using a hierarchical mode seeking and random forest method [7]. An accuracy of 87% was achieved for the training, and accuracy of 57% when testing new images. In 2013, the Karhunen-Loeve Transform was used to classify gesture images of one hand into 10 classes [8]. These were translated and the axes were rotated to distinguish a modern coordinate model by applying edge detection, hand cropping, and skin filter techniques. An accuracy of (96%) was achieved.

Sharma [9] characterized each colour channel after background deduction and noise elimination using (SVM and k-NN) classifiers, followed by a contour trace. An accuracy of (62.3%) was gained by using (SVM) as a classifier. Starner, Weaver & Pentland tracked hand movements by using a (3D) glove and an (HMM) model. This model can gain (3D) information from the hands regardless of the spatial direction. An accuracy of (99.2%) was achieved on the

test dataset. HMM utilizes time series datasets to follow hands movement and recognize them according to where the hand has been [10]. All the research that has been discussed above used linear classifiers, which are relatively simple and only require attribute extraction and pre-processing to be successful.

Another approach is to use deep learning techniques. This approach was used to build a model that recognizes hand gestures in a continual video stream utilizing DBN models [11]. An accuracy of over 99% was achieved. Another research used a deep learning technique, whereby a feed forward neural network was used to classify a sign. Many image pre-processing methods have been used, such as background subtraction, image normalization, image segmentation and contrast adjustment. In addition, a principal component analysis (PCA) and Gabor filters have been used for feature extraction. An accuracy of 98.5% was achieved with this method [12]. All the works discussed above depended on the extraction of the hand before it is fed to a network. However, a research was done on different sign languages from different countries [13], and this was the most relevant work for the current study. An Italian Sign Language recognition system was built using CNNs to classify 20 Italian gestures. A Microsoft Kinect was applied to full body images of people, whereby the Kinect was able to capture depth images. Only the depth images were used for training and an accuracy of 91.7% was achieved. However, it was mentioned that the test dataset could be in the training dataset (and/or) the validation dataset [13]. The structural design of the system consisted of two convolutional neural networks, one to extract higher body features and one to extract hand features. The data set, looking at People 2014, was used [14]. The depth map images were also used with a data set involving 20 Italian sign motions. The validation was 0.789675, and the final was 0.788804.

## **3. SYSTEM ANALYSIS**



### **3. SYSTEM ANALYSIS**

#### **SYSTEM ANALYSIS**

System Analysis is the important phase in the system development process. The System is studied to the minute details and analyzed. The system analyst plays an important role of an interrogator and dwells deep into the working of the present system. In analysis, a detailed study of these operations performed by the system and their relationships within and outside the system is done. A key question considered here is, “What must be done to solve the problem?”. The system is viewed as a whole and the inputs to the system are identified. Once analysis is completed the analyst has a firm understanding of what is to be done.

#### **3.1 PROBLEM DEFINITION**

A detailed study of the process must be made by various techniques like interviews, questionnaires etc. The data collected by these sources must be scrutinized to arrive at a conclusion. The conclusion is an understanding of how the system functions. This system is called the existing system. Now the existing system is subjected to close study and problem areas are identified. The designer now functions as a problem solver and tries to sort out the difficulties that the user faces. The model is built in such a way that it addresses the difficulties faced by the user. Consider the example, there are two users, the first one who needs a model which also allows him to move around in public and the second one who needs a model to be trained only to function inside his house, based on these requirements a model is generated and tested to see if the user is satisfied with the application.

#### **3.2 EXISTING SYSTEM**

We have some systems, the first one being the age-old Translators and we also have a system where the sign language can be converted to text, but this only is done for singular alphabets. Translators are not available in abundance thus making it very expensive to have one

available at all times, and the existing digital translators are very slow since every alphabet has to be gestured out and the amount of time it would take to just form a simple sentence would be a lot.

### 3.2.1 LIMITATIONS OF EXISTING SYSTEM

- These existing technologies are very restricted.
- They provide translations only on a particular standard.
- They can be very expensive.

To avoid all these limitations and make the working more accurate the system needs to be implemented efficiently.

## 3.3 PROPOSED SYSTEM

Initially we take pictures of the gesture made, once the pictures have been taken we use the LabelImg software to segregate these images into the appropriate labels. Once the labelling of the images has been done we have two sets of files for each image taken, one which has the actual image in it and the other being an XML file which contains information of where the model should be looking in the image during the training process. The model is trained on a **Deep Learning SSD** ML algorithm to extract features from the desired image. To achieve the detection we are using the **TensorFlow Object Detection** module, here the extracted features from the images taken are passed onto the TensorFlow module which is going to make comparisons with the real time video present in the frame. On detection of any of these features it is going to generate a prediction of the meaning which is the same as the name of the label.

### 3.3.1 ADVANTAGES OF THE PROPOSED SYSTEM

The system is very simple in design and to implement. The system requires very low system resources and the system will work in almost all configurations. It has the following features.

- The main Advantage is that this model is much more efficient than the existing technologies.
- The system allows the user to train the model as per their set standards.
- The user can provide the images of the gestures and also provide the meaning of the gestures for labelling to improve the accuracy of the predictions made by the model.

### **3.4 FEASIBILITY STUDY**

The feasibility of the project is analyzed in this phase and a business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis the feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the user.

Three key considerations involved in the feasibility analysis are

- Economic Feasibility
- Technical Feasibility
- Social Feasibility

#### **3.4.1 ECONOMIC FEASIBILITY**

The developing system must be justified by cost and benefit. Criteria to ensure that effort is concentrated on a project, which will give the user the best quality of life possible. One of the factors, which affect the development of a new system, is the cost it would require.

The following are some of the important financial questions asked during preliminary investigation:

- The costs conduct a full system investigation.
- The cost of the hardware and software.
- The benefits in the form of reduced costs or fewer costly errors.

Since the system is developed as part of project work, there is no manual cost to spend for the proposed system. Also, all the resources are already available, it gives an indication that the system is economically possible for development.

### **3.4.2 TECHNICAL FEASIBILITY**

This study is carried out to check the technical feasibility, that is, the technical requirements of the system. Any system developed must not have a high demand on the available technical resources. The developed system must have a modest requirement, as only minimal or null changes are required for implementing this system.

### **3.4.3 BEHAVIORAL FEASIBILITY**

This includes the following questions:

- Is there sufficient support for the users?
- Will the proposed system cause harm?

The project would be beneficial because it satisfies the objectives when developed and installed. All behavioral aspects are considered carefully and conclude that the project is behaviorally feasible.

## **3.5 HARDWARE & SOFTWARE REQUIREMENTS**

### **3.5.1 HARDWARE REQUIREMENTS**

Hardware interfaces specify the logical characteristics of each interface between the software product and the hardware components of the system. The following are some hardware requirements.

- Processor : Pentium V or higher
- RAM : 4 GB or higher
- Space on Hard Disk : Minimum 10 GB
- High Quality Camera

### 3.5.2 SOFTWARE REQUIREMENTS

Software Requirements specifies the logical characteristics of each interface and software components of the system. The following are some software requirements:

- Operating System : Windows 10, Windows 8
- Technologies : TensorFlow, OpenCV, Deep Learning SSD Module
- Language : Python
- Jupyter Notebook
- LabelImg Software

# **4. ARCHITECTURE**

## 4. ARCHITECTURE

### 4.1 PROJECT ARCHITECTURE

This project architecture describes how the application is going to function. The detailed architecture is explained below.

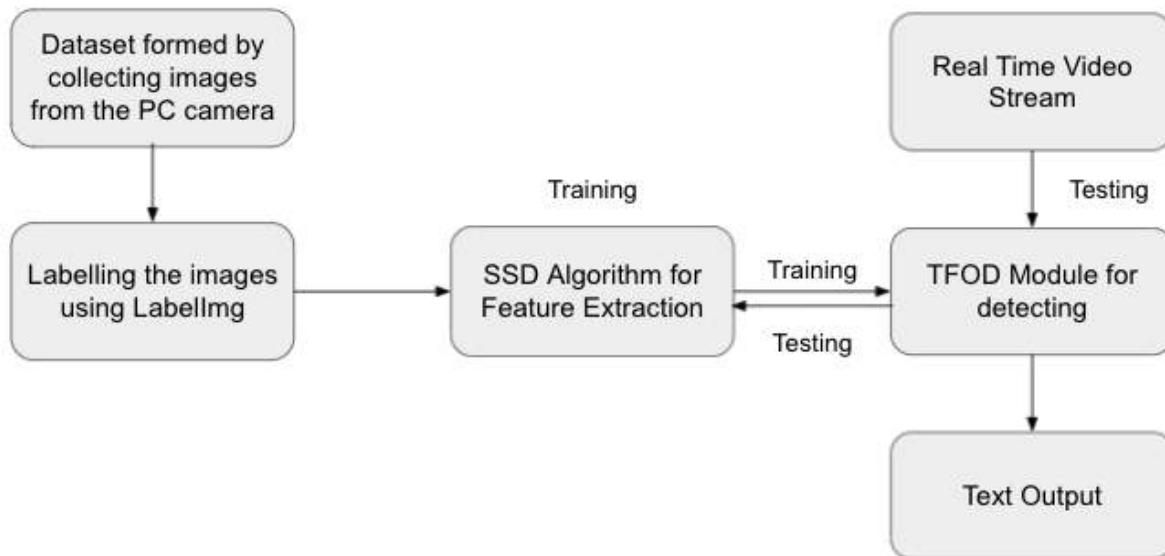


Figure 4.1: Project Architecture for ML Based Real Time Sign Language Detection

### 4.2 DESCRIPTION

- **Dataset formed by Collecting Images from the PC Camera:** A set of 20 images for each gesture/sign were taken using the PC Camera for training as well as testing the ML model.
- **Labelling the Images using Labellmg:** Each image collected above was labelled using the Labellmg software. The Labellmg software is used for graphically labelling the images that is further used when recognizing the images. It is important to understand the gesture made and give the correct label to it, as a wrong label can lead to miscommunication.

- **SSD Algorithm for Feature Extraction:** The ML model was trained using the Deep Learning SSD ML Algorithm. SSD ( Single Shot Detection) algorithm is designed for object detection in real-time. The SSD architecture is a single convolution network that learns to predict bounding box locations and classify these locations in one pass. Hence, SSD can be trained end-to-end.
- **TFOD Module for Detecting:** The ML model was tested using the TensorFlow Object Detection API. The TensorFlow Object Detection API is an open-source framework built on top of TensorFlow that makes it easy to construct, train and deploy object detection models. TensorFlow bundles together Machine Learning and Deep Learning models and algorithms.
- **Real Time Video Stream:** A real time video stream for the gestures being made which is captured using the PC Camera is given as an input to the ML model to know what gestures are being made.
- **Text Output:** The meaning of the gestures being made is shown in text form with a bounding box around the gesture. This text is the same as the label given to the gesture/sign while labelling the images using LabelImg.



### 4.3 USE CASE DIAGRAM

In the use case diagram, we basically have four actors, namely: the User, TensorFlow, Labelling and the Database. The user has the following methods, set labels and show gestures. The TensorFlow module has only one method, that is, gesture recognition. The Labelling software has the following methods, set labels and display labels. The database has only one method, that is, get gestures.

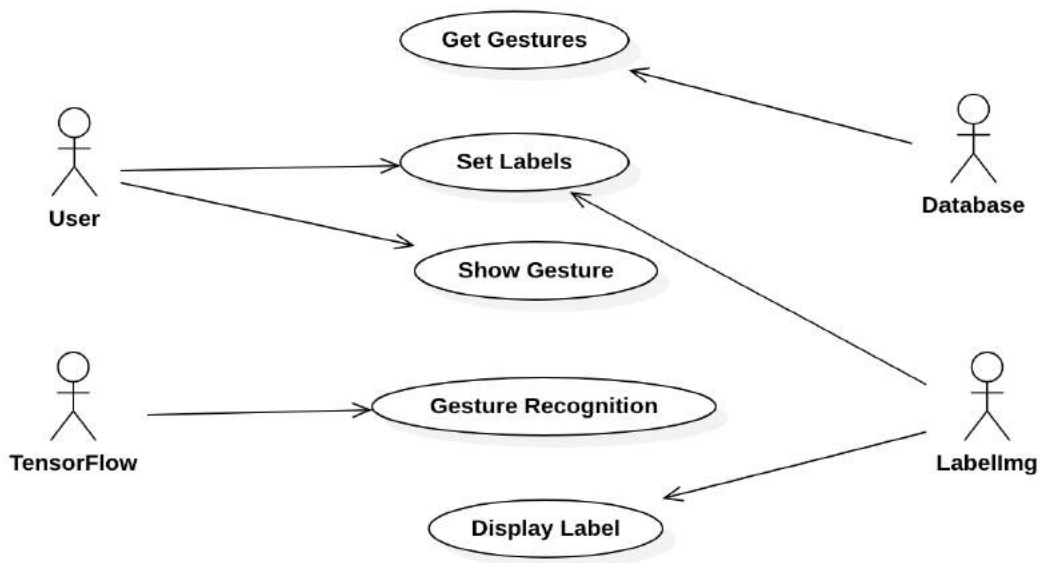


Figure 4.2: Use Case Diagram for ML Based Real Time Sign Language Detection

## 4.4 CLASS DIAGRAM

Class Diagram is a collection of classes and objects.

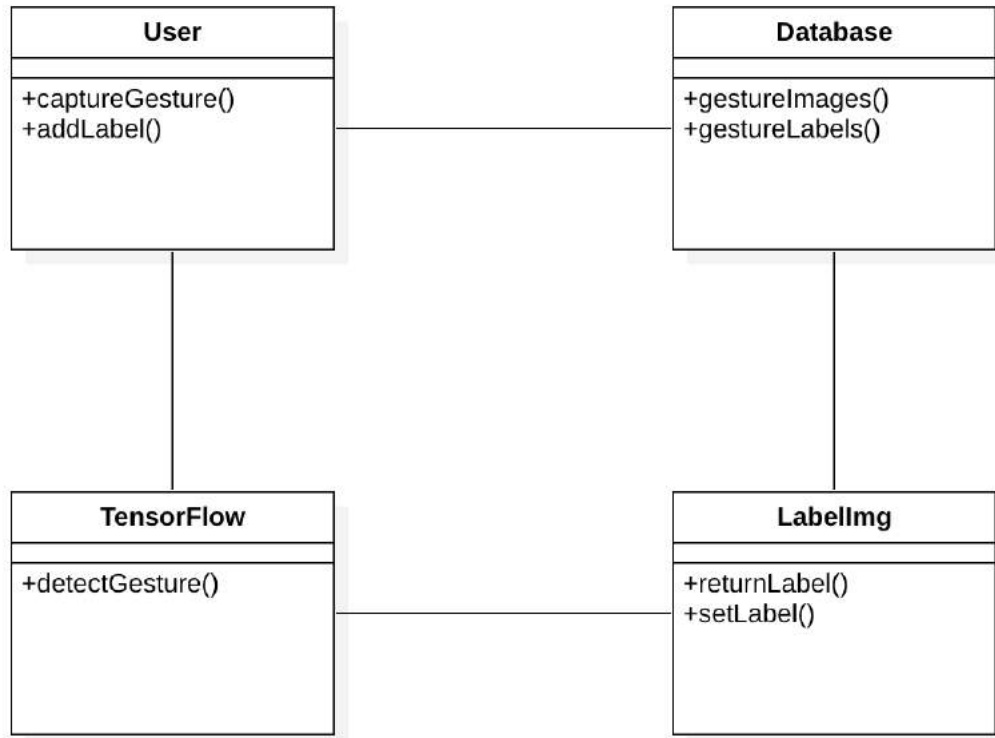


Figure 4.3: Class Diagram for ML Based Real Time Sign Language Detection

## 4.5 SEQUENCE DIAGRAM

The sequence diagram shows the sequence in which different tasks are being carried out by the actors.

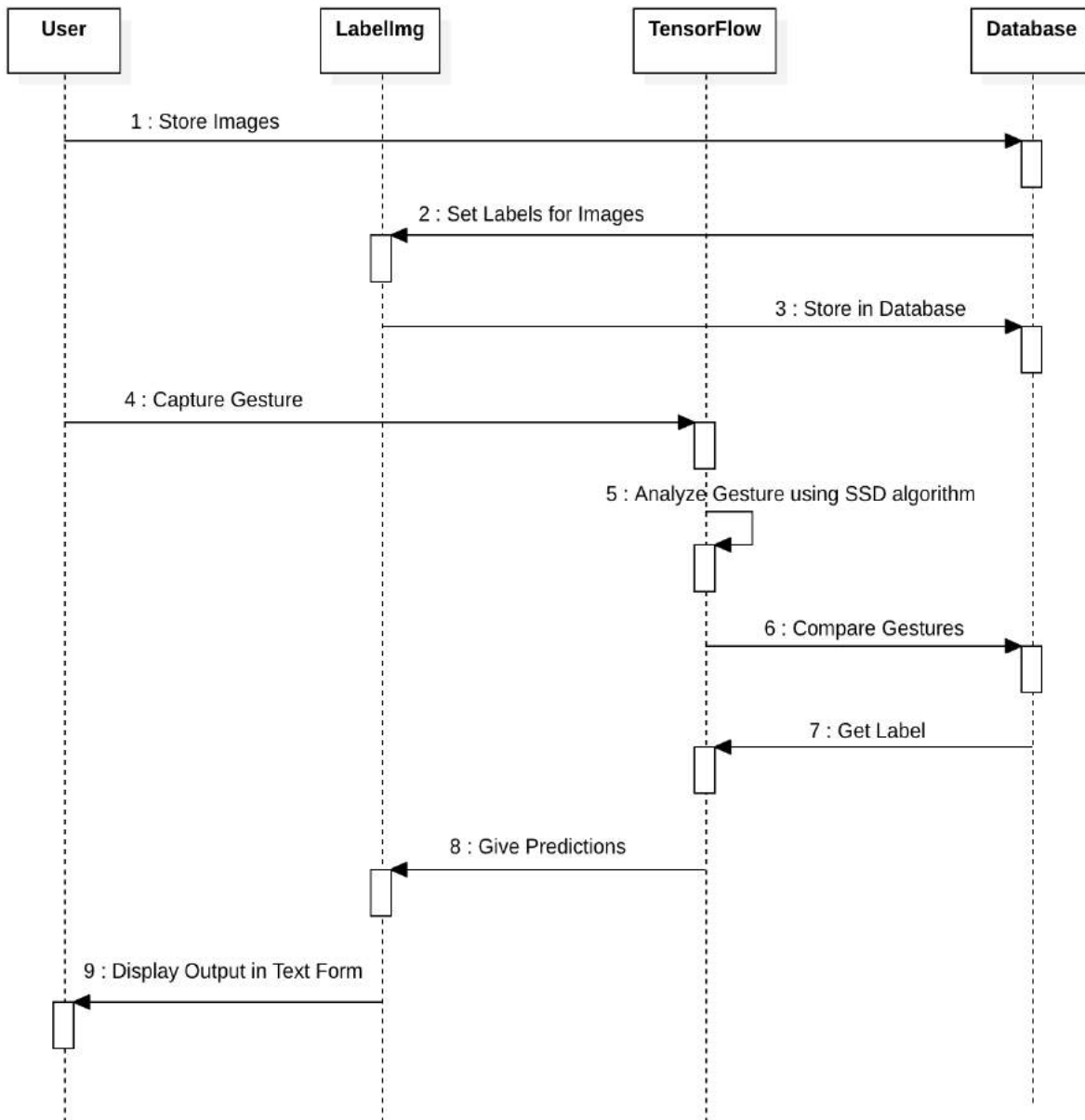


Figure 4.4: Sequence Diagram for ML Based Real Time Sign Language Detection

## 4.6 ACTIVITY DIAGRAM

Activity Diagram describes the flow of activity states.

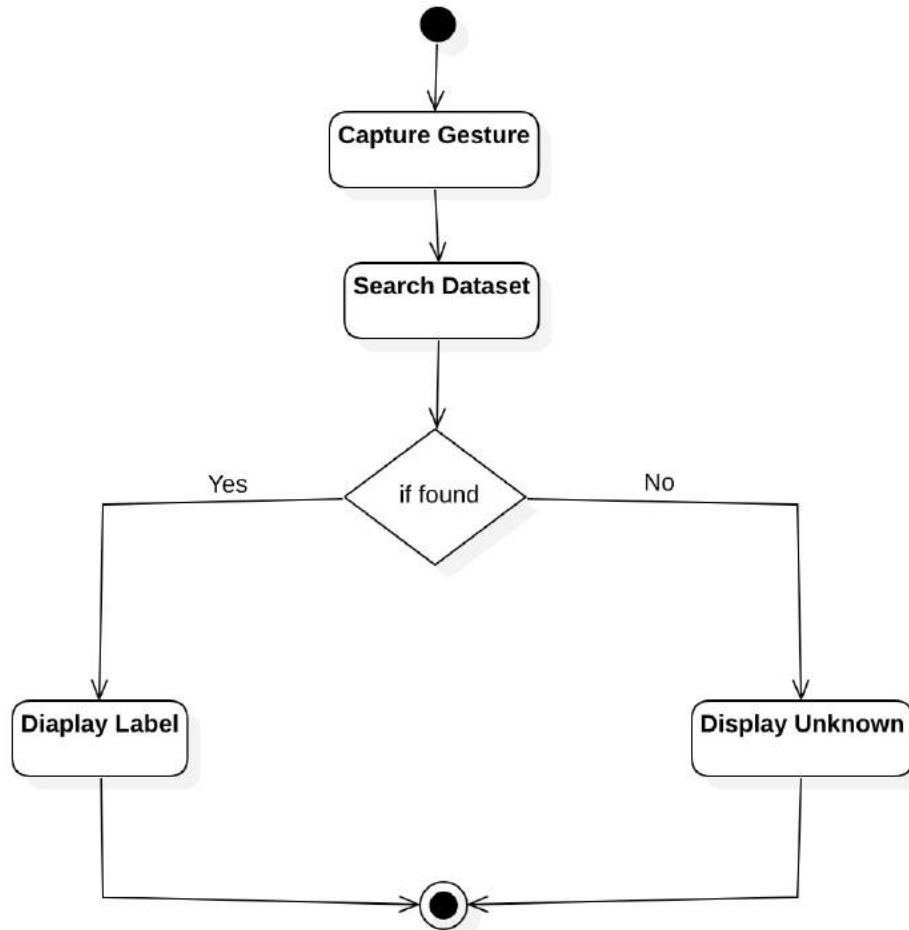


Figure 4.5: Activity Diagram for ML Based Real Time Sign Language Detection

# **5. IMPLEMENTATION**

## 5. IMPLEMENTATION

### 5.1 SAMPLE CODE

#### #Creation and Activation of Virtual Environment

```
python -m venv tfod
.\tfod\Scripts\activate
```

#### #Installing Dependencies and Adding Virtual Environment to Jupyter Notebook

```
python -m pip install --upgrade pip
pip install ipykernel
python -m ipykernel install --user --name=tfod
```

#### # Automated Image Collection for Dataset

```
!pip install opencv-python
import cv2          #Import opencv
import uuid         #Import uuid
import os           #Import Operating System
import time        #Import Time

labels = ['Hello', 'ThankYou', 'Yes', 'No', 'ILoveYou'] # Label Creation
number_imgs = 20   #Number of Images to be taken
IMAGES_PATH = os.path.join('Tensorflow', 'workspace', 'images', 'collectedimages')
if not os.path.exists(IMAGES_PATH):
    !mkdir {IMAGES_PATH}
for label in labels:
    path = os.path.join(IMAGES_PATH, label)
    if not os.path.exists(path):
        !mkdir {path}
```

**#Image Collection**

```

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(IMGES_PATH,label,label+'.'+{}.format(str(uuid.uuid1())))
        cv2.imwrite(imgname, frame)
        cv2.imshow('frame', frame)
        time.sleep(2)

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

```

**# Installing Dependencies for LabelImg**

```

!pip install --upgrade pyqt5 lxml
LABELIMG_PATH = os.path.join('Tensorflow', 'labelimg')
if not os.path.exists(LABELIMG_PATH):
    !mkdir {LABELIMG_PATH}
    !git clone https://github.com/tzutalin/labelImg {LABELIMG_PATH}
!cd {LABELIMG_PATH} && pyrcc5 -o libs/resources.py resources.qrc

```

```
!cd {LABELIMG_PATH} && python labelImg.py
```

### # Training and Detection Module:

```
import os

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\_coco17\_tpu-8.tar.gz'

TF_RECORD_SCRIPT_NAME = 'generate_tfrecord.py'

LABEL_MAP_NAME = 'label_map.pbtxt'

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')
}
```



```

files = {
    'PIPELINE_CONFIG': os.path.join('Tensorflow', 'workspace', 'models',
CUSTOM_MODEL_NAME, 'pipeline.config'),
    'TF_RECORD_SCRIPT': os.path.join(paths['SCRIPTS_PATH'],
TF_RECORD_SCRIPT_NAME),
    'LABELMAP': os.path.join(paths['ANNOTATION_PATH'], LABEL_MAP_NAME)
}
for path in paths.values():
    if not os.path.exists(path):
        !mkdir {path}
!pip install wget
import wget
if not os.path.exists(os.path.join(paths['APIMODEL_PATH'], 'research', 'object_detection')):
    !git clone https://github.com/tensorflow/models {paths['APIMODEL_PATH']}

# Installing TensorFlow
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\tf2\setup.py setup.py && python setup.py build &&
python setup.py install

```

```
!cd Tensorflow/models/research/slim && pip install -e .
```

### # Verification Script For Tensorflow Installation

```
VERIFICATION_SCRIPT = os.path.join(paths['APIMODEL_PATH'], 'research',
'object_detection', 'builders', 'model_builder_tf2_test.py')
```

```
!python {VERIFICATION_SCRIPT}
```

### # Training Process

```
import object_detection
```

```
wget.download(PRETRAINED_MODEL_URL)
```

```
!move PRETRAINED_MODEL_NAME+'.tar.gz' {paths['PRETRAINED_MODEL_PATH']}
```

```
!cd {paths['PRETRAINED_MODEL_PATH']} &&
tar-zxvf{PRETRAINED_MODEL_NAME+'.tar.gz'}
```

```
labels = [{'name':'Hello', 'id':1}, {'name':'ThankYou', 'id':2}, {'name':'Yes', 'id':3}, {'name':'No',
'id':4}, {'name':'ILoveYou', 'id':5}]
```

```
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')
```

```
if not os.path.exists(files['TF_RECORD_SCRIPT']):
```

```
    !git clone https://github.com/nicknochnack/GenerateTFRecord {paths['SCRIPTS_PATH']}
```

```
    !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')}
```

```
    !copy {os.path.join(paths['PRETRAINED_MODEL_PATH'],
PRETRAINED_MODEL_NAME, 'pipeline.config')}
{os.path.join(paths['CHECKPOINT_PATH'])}
```

```

import tensorflow as tf

from object_detection.utils import config_util

from object_detection.protos import pipeline_pb2

from google.protobuf import text_format

config = config_util.get_configs_from_pipeline_file(files['PIPELINE_CONFIG'])

pipeline_config = pipeline_pb2.TrainEvalPipelineConfig()

with tf.io.gfile.GFile(files['PIPELINE_CONFIG'], "r") as f:

    proto_str = f.read()

    text_format.Merge(proto_str, pipeline_config)

pipeline_config.model.ssd.num_classes = len(labels)

pipeline_config.train_config.batch_size = 4

pipeline_config.train_config.fine_tune_checkpoint =
os.path.join(paths['PRETRAINED_MODEL_PATH'], PRETRAINED_MODEL_NAME,
'checkpoint', 'ckpt-0')

pipeline_config.train_config.fine_tune_checkpoint_type = "detection"

pipeline_config.train_input_reader.label_map_path= files['LABELMAP']

pipeline_config.train_input_reader.tf_record_input_reader.input_path[:] =
[os.path.join(paths['ANNOTATION_PATH'], 'train.record')]

pipeline_config.eval_input_reader[0].label_map_path = files['LABELMAP']

pipeline_config.eval_input_reader[0].tf_record_input_reader.input_path[:] =
[os.path.join(paths['ANNOTATION_PATH'], 'test.record')]

config_text = text_format.MessageToString(pipeline_config)

with tf.io.gfile.GFile(files['PIPELINE_CONFIG'], "wb") as f:

    f.write(config_text)

TRAINING_SCRIPT = os.path.join(paths['APIMODEL_PATH'], 'research', 'object_detection',
'model_main_tf2.py')

command = "python {} --model_dir={} --pipeline_config_path={}
--num_train_steps=2000".format(TRAINING_SCRIPT,
paths['CHECKPOINT_PATH'],files['PIPELINE_CONFIG'])

```

```
print(command)          # Copy the printed command and run in CMD of virtual environment
```

### **#Evaluation of the Model to know Loss Metrics and Confidence.**

```
command = "python {} --model_dir={} --pipeline_config_path={}
--checkpoint_dir={}".format(TRAINING_SCRIPT,
paths['CHECKPOINT_PATH'],files['PIPELINE_CONFIG'], paths['CHECKPOINT_PATH'])
```

```
print(command)          # Copy Command and run in CMD of Virtual Environment
```

### **#Load Model From Checkpoint**

```
import os

import tensorflow as tf

from object_detection.utils import label_map_util
from object_detection.utils import visualization_utils as viz_utils
from object_detection.builders import model_builder
from object_detection.utils import config_util

configs = config_util.get_configs_from_pipeline_file(files['PIPELINE_CONFIG'])
detection_model = model_builder.build(model_config=configs['model'], is_training=False)
ckpt = tf.compat.v2.train.Checkpoint(model=detection_model)
ckpt.restore(os.path.join(paths['CHECKPOINT_PATH'], 'ckpt-3')).expect_partial()

@tf.function
def detect_fn(image):
    image, shapes = detection_model.preprocess(image)
    prediction_dict = detection_model.predict(image, shapes)
    detections = detection_model.postprocess(prediction_dict, shapes)
    return detections

import cv2

import numpy as np

from matplotlib import pyplot as plt

%matplotlib inline

category_index = label_map_util.create_category_index_from_labelmap(files['LABELMAP'])
```

**# Detection In Real Time Using Webcam**

```

cap = cv2.VideoCapture(0)
width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
while cap.isOpened():
    ret, frame = cap.read()
    image_np = np.array(frame)
    input_tensor = tf.convert_to_tensor(np.expand_dims(image_np, 0), dtype=tf.float32)
    detections = detect_fn(input_tensor)
    num_detections = int(detections.pop('num_detections'))
    detections = {key: value[0, :num_detections].numpy()
                  for key, value in detections.items()}
    detections['num_detections'] = num_detections
    detections['detection_classes'] = detections['detection_classes'].astype(np.int64)
    label_id_offset = 1
    image_np_with_detections = image_np.copy()
    viz_utils.visualize_boxes_and_labels_on_image_array(
        image_np_with_detections,
        detections['detection_boxes'],
        detections['detection_classes']+label_id_offset,
        detections['detection_scores'],
        category_index,
        use_normalized_coordinates=True,
        max_boxes_to_draw=5,
        min_score_thresh=.7,
        agnostic_mode=False)

```

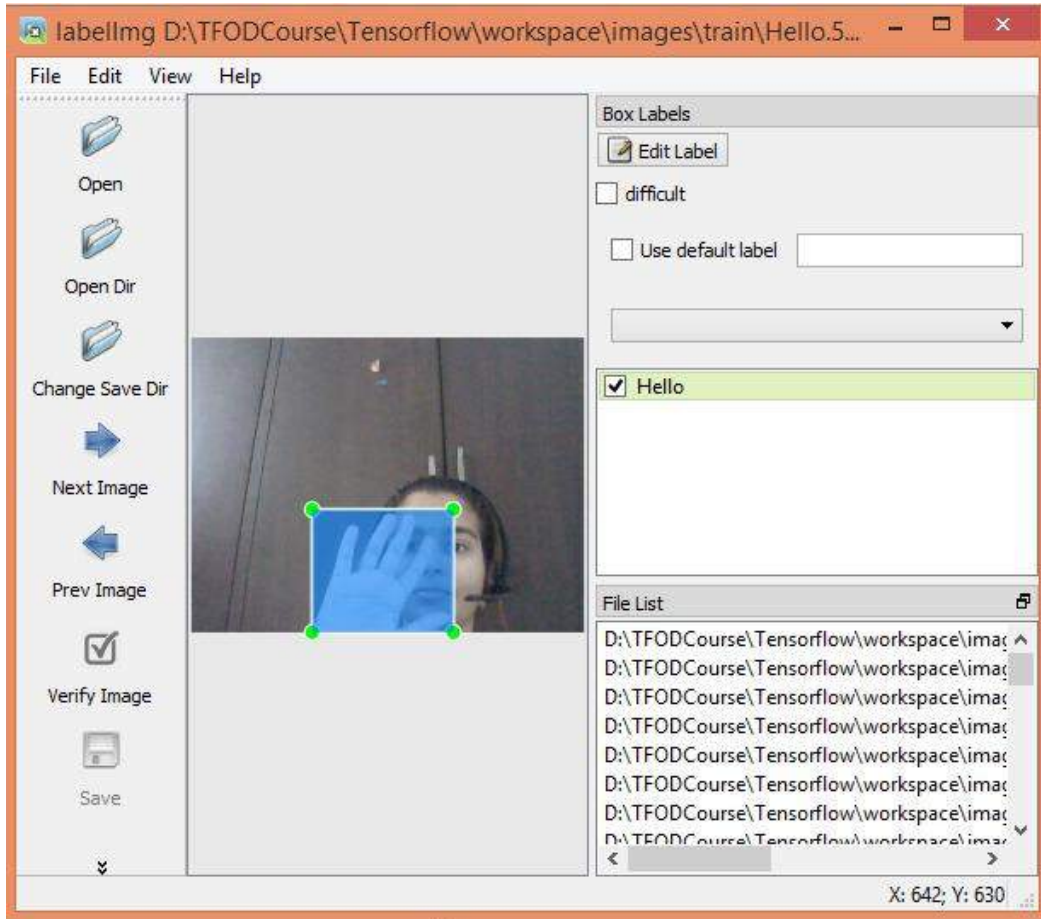
```
cv2.imshow('object detection', cv2.resize(image_np_with_detections, (800, 600)))  
if cv2.waitKey(10) & 0xFF == ord('q'):  
    cap.release()  
    cv2.destroyAllWindows()  
    break
```

## **6. SCREENSHOTS**

## 6. SCREENSHOTS

### 6.1 LABELLING IMAGES USING LABELIMG

This is how the images are labelled using the LabelImg software.



Screenshot 6.1: LabelImg Software



## 6.2 GRAYSCALE

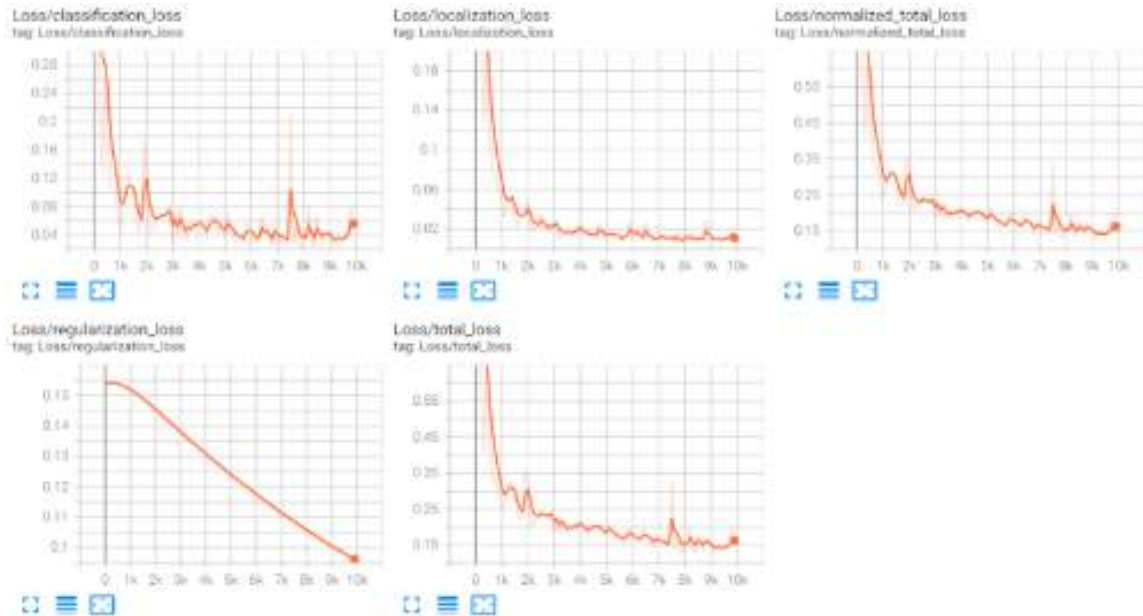
All the images were converted into gray scale images as shown in Screenshot 6.2., for training the ML model.



Screenshot 6.2: Images for Training in Grayscale

### 6.3 LOSS OF MACHINE LEARNING MODEL - GRAPH

The loss at each iteration of our machine learning model has been decreasing which indicates a better accuracy of model for detection. The loss of our model is shown in the below Screenshot 6.3.



Screenshot 6.3: Loss of the Machine Learning Model

## 6.4 LOSS OF MACHINE LEARNING MODEL AT EACH STEP

The below screenshot shows the loss incurred at each step while training the model. The lowest loss recorded is 0.133 at step 9400 and at step 9700.

```

Command Prompt
INFO:tensorflow:Step 7700 per-step time 0.174s loss=0.161
I0515 16:54:50.780918 16348 model_lib_v2.py:683] Step 7700 per-step time 0.174s loss=0.161
INFO:tensorflow:Step 7800 per-step time 0.173s loss=0.137
I0515 16:55:08.102036 16348 model_lib_v2.py:683] Step 7800 per-step time 0.173s loss=0.137
INFO:tensorflow:Step 7900 per-step time 0.174s loss=0.197
I0515 16:55:25.463247 16348 model_lib_v2.py:683] Step 7900 per-step time 0.174s loss=0.197
INFO:tensorflow:Step 8000 per-step time 0.173s loss=0.166
I0515 16:55:42.735772 16348 model_lib_v2.py:683] Step 8000 per-step time 0.173s loss=0.166
INFO:tensorflow:Step 8100 per-step time 0.181s loss=0.152
I0515 16:56:00.805616 16348 model_lib_v2.py:683] Step 8100 per-step time 0.181s loss=0.152
INFO:tensorflow:Step 8200 per-step time 0.177s loss=0.144
I0515 16:56:18.493315 16348 model_lib_v2.py:683] Step 8200 per-step time 0.177s loss=0.144
INFO:tensorflow:Step 8300 per-step time 0.176s loss=0.136
I0515 16:56:36.113846 16348 model_lib_v2.py:683] Step 8300 per-step time 0.176s loss=0.136
INFO:tensorflow:Step 8400 per-step time 0.179s loss=0.154
I0515 16:56:54.052024 16348 model_lib_v2.py:683] Step 8400 per-step time 0.179s loss=0.154
INFO:tensorflow:Step 8500 per-step time 0.180s loss=0.165
I0515 16:57:12.006381 16348 model_lib_v2.py:683] Step 8500 per-step time 0.180s loss=0.165
INFO:tensorflow:Step 8600 per-step time 0.179s loss=0.146
I0515 16:57:29.890624 16348 model_lib_v2.py:683] Step 8600 per-step time 0.179s loss=0.146
INFO:tensorflow:Step 8700 per-step time 0.180s loss=0.134
I0515 16:57:47.875270 16348 model_lib_v2.py:683] Step 8700 per-step time 0.180s loss=0.134
INFO:tensorflow:Step 8800 per-step time 0.181s loss=0.144
I0515 16:58:05.941070 16348 model_lib_v2.py:683] Step 8800 per-step time 0.181s loss=0.144
INFO:tensorflow:Step 8900 per-step time 0.182s loss=0.155
I0515 16:58:24.166033 16348 model_lib_v2.py:683] Step 8900 per-step time 0.182s loss=0.155
INFO:tensorflow:Step 9000 per-step time 0.180s loss=0.176
I0515 16:58:42.164034 16348 model_lib_v2.py:683] Step 9000 per-step time 0.180s loss=0.176
INFO:tensorflow:Step 9100 per-step time 0.190s loss=0.181
I0515 16:59:01.203125 16348 model_lib_v2.py:683] Step 9100 per-step time 0.190s loss=0.181
INFO:tensorflow:Step 9200 per-step time 0.180s loss=0.180
I0515 16:59:19.224931 16348 model_lib_v2.py:683] Step 9200 per-step time 0.180s loss=0.180
INFO:tensorflow:Step 9300 per-step time 0.180s loss=0.143
I0515 16:59:37.192048 16348 model_lib_v2.py:683] Step 9300 per-step time 0.180s loss=0.143
INFO:tensorflow:Step 9400 per-step time 0.178s loss=0.133
I0515 16:59:55.041156 16348 model_lib_v2.py:683] Step 9400 per-step time 0.178s loss=0.133
INFO:tensorflow:Step 9500 per-step time 0.177s loss=0.172
I0515 17:00:12.711158 16348 model_lib_v2.py:683] Step 9500 per-step time 0.177s loss=0.172
INFO:tensorflow:Step 9600 per-step time 0.180s loss=0.162
I0515 17:00:30.676843 16348 model_lib_v2.py:683] Step 9600 per-step time 0.180s loss=0.162
INFO:tensorflow:Step 9700 per-step time 0.180s loss=0.133
I0515 17:00:48.735874 16348 model_lib_v2.py:683] Step 9700 per-step time 0.180s loss=0.133
INFO:tensorflow:Step 9800 per-step time 0.180s loss=0.140
I0515 17:01:06.709062 16348 model_lib_v2.py:683] Step 9800 per-step time 0.180s loss=0.140
INFO:tensorflow:Step 9900 per-step time 0.179s loss=0.137
I0515 17:01:24.646522 16348 model_lib_v2.py:683] Step 9900 per-step time 0.179s loss=0.137
INFO:tensorflow:Step 10000 per-step time 0.179s loss=0.135
I0515 17:01:42.572833 16348 model_lib_v2.py:683] Step 10000 per-step time 0.179s loss=0.135

```

Screenshot 6.4: Loss at each Iteration



## 6.5 EVALUATION METRIC

The below screenshot represents the evaluation results and evaluation metrics for a 10000 step machine learning model. An evaluation metric consists of the average precision and average recall.

```

Accumulating evaluation results...
DONE (t=0.03s).
Average Precision (AP) @[ IoU=0.50:0.95 | area= all | maxDets=100 ] = 0.785
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 ] = -1.000
Average Precision (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.785
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets= 1 ] = 0.795
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets= 10 ] = 0.795
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets=100 ] = 0.795
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 ] = -1.000
Average Recall (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.795
INFO:tensorflow:Eval metrics at step 2000
I0515 16:36:29.779896 18012 model_lib_v2.py:989] Eval metrics at step 2000
INFO:tensorflow: + DetectionBoxes_Precision/mAP: 0.784736
I0515 16:36:29.826962 18012 model_lib_v2.py:992] + DetectionBoxes_Precision/mAP: 0.784736
INFO:tensorflow: + DetectionBoxes_Precision/mAP@.50IOU: 1.000000
I0515 16:36:29.828956 18012 model_lib_v2.py:992] + DetectionBoxes_Precision/mAP@.50IOU: 1.000000
INFO:tensorflow: + DetectionBoxes_Precision/mAP@.75IOU: 1.000000
I0515 16:36:29.830981 18012 model_lib_v2.py:992] + DetectionBoxes_Precision/mAP@.75IOU: 1.000000
INFO:tensorflow: + DetectionBoxes_Precision/mAP (small): -1.000000
I0515 16:36:29.831947 18012 model_lib_v2.py:992] + DetectionBoxes_Precision/mAP (small): -1.000000
INFO:tensorflow: + DetectionBoxes_Precision/mAP (medium): -1.000000
I0515 16:36:29.832945 18012 model_lib_v2.py:992] + DetectionBoxes_Precision/mAP (medium): -1.000000
INFO:tensorflow: + DetectionBoxes_Precision/mAP (large): 0.784736
I0515 16:36:29.835937 18012 model_lib_v2.py:992] + DetectionBoxes_Precision/mAP (large): 0.784736
INFO:tensorflow: + DetectionBoxes_Recall/AR@1: 0.795000
I0515 16:36:29.837959 18012 model_lib_v2.py:992] + DetectionBoxes_Recall/AR@1: 0.795000
INFO:tensorflow: + DetectionBoxes_Recall/AR@10: 0.795000
I0515 16:36:29.838930 18012 model_lib_v2.py:992] + DetectionBoxes_Recall/AR@10: 0.795000
INFO:tensorflow: + DetectionBoxes_Recall/AR@100: 0.795000
I0515 16:36:29.841921 18012 model_lib_v2.py:992] + DetectionBoxes_Recall/AR@100: 0.795000
INFO:tensorflow: + DetectionBoxes_Recall/AR@100 (small): -1.000000
I0515 16:36:29.852892 18012 model_lib_v2.py:992] + DetectionBoxes_Recall/AR@100 (small): -1.000000
INFO:tensorflow: + DetectionBoxes_Recall/AR@100 (medium): -1.000000
I0515 16:36:29.853890 18012 model_lib_v2.py:992] + DetectionBoxes_Recall/AR@100 (medium): -1.000000
INFO:tensorflow: + DetectionBoxes_Recall/AR@100 (large): 0.795000
I0515 16:36:29.884561 18012 model_lib_v2.py:992] + DetectionBoxes_Recall/AR@100 (large): 0.795000
INFO:tensorflow: + Loss/localization_loss: 0.059917
I0515 16:36:29.888390 18012 model_lib_v2.py:992] + Loss/localization_loss: 0.059917
INFO:tensorflow: + Loss/classification_loss: 0.192864
I0515 16:36:29.895365 18012 model_lib_v2.py:992] + Loss/classification_loss: 0.192864
INFO:tensorflow: + Loss/regularization_loss: 0.144596
I0515 16:36:29.897387 18012 model_lib_v2.py:992] + Loss/regularization_loss: 0.144596
INFO:tensorflow: + Loss/total_loss: 0.397377
I0515 16:36:29.901349 18012 model_lib_v2.py:992] + Loss/total_loss: 0.397377

```

Screenshot 6.5: Evaluation Results and Evaluation Metrics

## 6.6 SINGLE GESTURE RECOGNITION

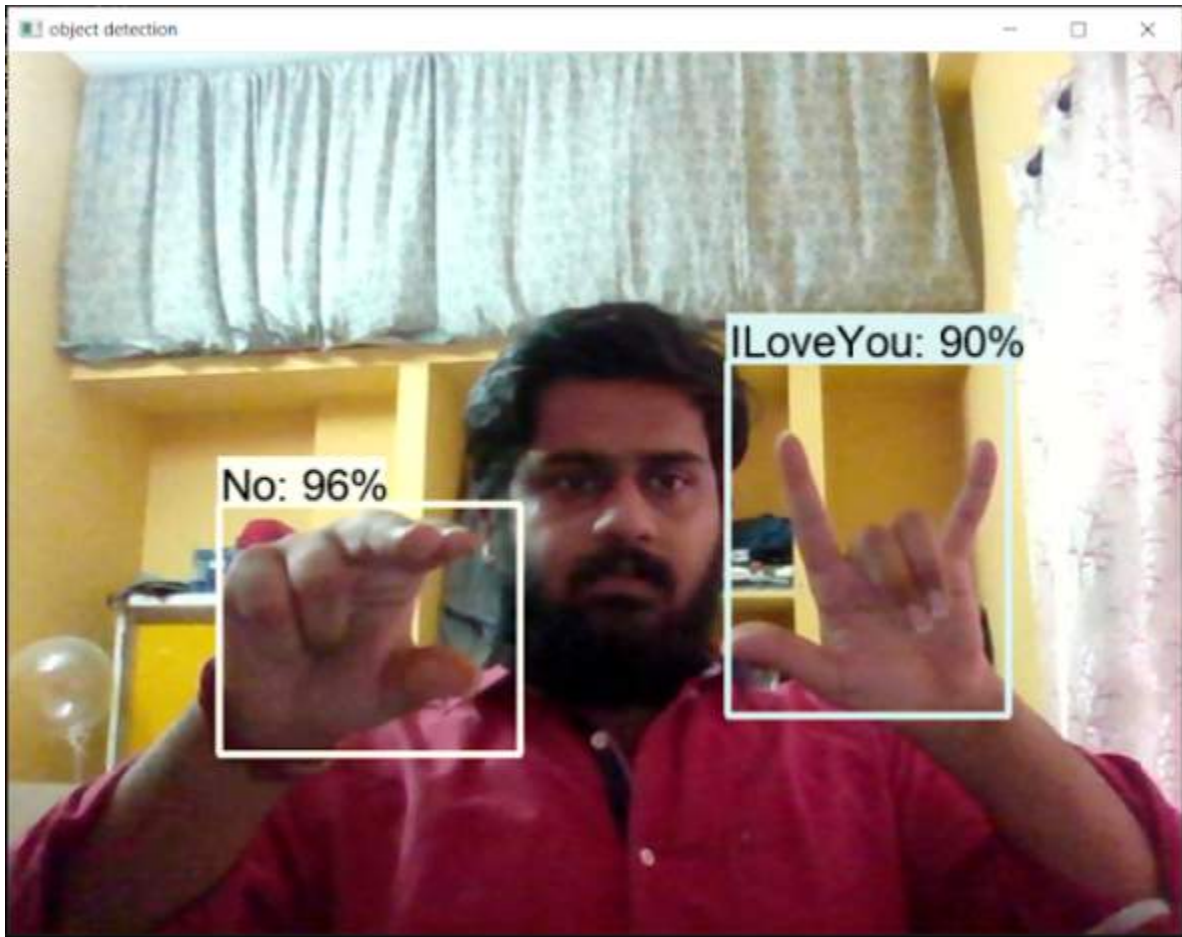
Screenshot 6.6 shows the recognition of the sign 'No' by the ML model with an accuracy of 90%.



Screenshot 6.6: Gesture Recognition for No

## 6.7 TWO GESTURE RECOGNITION

Two gestures can also be recognised simultaneously. In the below screenshot we can see that both the gestures are being recognised without any difficulty.



Screenshot 6.7: Gesture Recognition for I Love You and No

# **7. TESTING**

## **7. TESTING**

### **7.1 INTRODUCTION TO TESTING**

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, subassemblies, assemblies and/or a finished product. It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of tests. Each test type addresses a specific testing requirement.

### **7.2 TYPES OF TESTING**

#### **7.2.1 UNIT TESTING**

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application. It is done after the completion of an individual unit before integration. This is a structural testing that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

#### **7.2.2 INTEGRATION TESTING**

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfactory, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.



### 7.2.3 FUNCTIONAL TESTING

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items:

Valid Input : identified classes of valid input must be accepted.

Invalid Input : identified classes of invalid input must be rejected.

Functions : identified functions must be exercised.

Output : identified classes of application outputs must be exercised.

Systems/Procedures: interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identifying Business process flows; data fields, predefined processes.

## 7.3 TEST CASES

### 7.3.1 OBJECT DETECTION

Test Case ID	Test Case Name	Purpose	Test Case	Output
1	Object detection	To check if the gesture is being recognized or not.	A video stream consisting of a Person making a gesture “ <b>No</b> ” is given as input.	A text feedback “ <b>No</b> ” with a bounding box around the gesture is given as output.
2	Object detection	To check if the gesture is being recognized or not.	A video stream consisting of a Person making a gesture “ <b>No</b> ” and “ <b>ILoveYou</b> ” is given as input simultaneously.	A text feedback “ <b>No</b> ” and “ <b>ILoveYou</b> ” with a bounding box around the gesture is given as output.

## **8. CONCLUSION**

## **8. CONCLUSION & FUTURE ENHANCEMENTS**

### **8.1 PROJECT CONCLUSION**

Through this project, we aim at providing a platform for the differently abled people to communicate with others irrespective of whether they know sign language or not. The interface of the platform is very simple making it very easy to use. Finally, we would like to say that we aim at diminishing the boundaries created by various medical conditions that restrict differently abled people to communicate with others and improve their quality of life.

### **8.2 FUTURE ENHANCEMENTS**

There is a lot of scope for this project, since we have the ability to label these images, we can label the gestures in any language required, allowing the user to communicate with others irrespective of the language boundaries.

## **9. BIBLIOGRAPHY**

## 9. BIBLIOGRAPHY

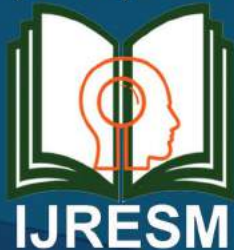
### 9.1 REFERENCES

- [1]. Garcia, B., & Viesca, S. A. (2016). Real-time American sign language recognition with convolutional neural networks. *Convolutional Neural Networks for Visual Recognition*, 2, 225-232.
- [2]. Van den Bergh, M., & Van Gool, L. (2011, January). Combining RGB and ToF cameras for real-time 3D hand gesture interaction. In *2011 IEEE workshop on applications of computer vision (WACV)* (pp. 66-72). IEEE.
- [3]. Liwicki, S., & Everingham, M. (2009, June). Automatic recognition of fingerspelled words in british sign language. In *2009 IEEE computer society conference on computer vision and pattern recognition workshops* (pp. 50-57). IEEE.
- [4]. Zafrulla, Z., Brashear, H., Starner, T., Hamilton, H., & Presti, P. (2011, November). American sign language recognition with the kinect. In *Proceedings of the 13th international conference on multimodal interfaces* (pp. 279-286).
- [5]. Pugeault, N., & Bowden, R. (2011, November). Spelling it out: Real-time ASL fingerspelling recognition. In *2011 IEEE International conference on computer vision workshops (ICCV workshops)* (pp. 1114-1119). IEEE.
- [6]. Kuznetsova, A., Leal-Taixé, L., & Rosenhahn, B. (2013). Real-time sign language recognition using a consumer depth camera. In *Proceedings of the IEEE International Conference on Computer Vision Workshops* (pp. 83-90).
- [7]. Dong, C., Leu, M. C., & Yin, Z. (2015). American sign language alphabet recognition using microsoft kinect. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops* (pp. 44-52).
- [8]. Singha, J., & Das, K. (2013). Hand gesture recognition based on Karhunen-Loeve transform. *arXiv preprint arXiv:1306.2599*.

- [9]. Sharma, R., Nemani, Y., Kumar, S., Kane, L., & Khanna, P. (2013, July). Recognition of single handed sign language gestures using contour tracing descriptor. In Proceedings of the World Congress on Engineering (Vol. 2, pp. 3-5).
- [10]. Starner, T., Weaver, J., & Pentland, A. (1998). Realtime american sign language recognition using desk and wearable computer based video. IEEE Transactions on pattern analysis and machine intelligence, 20(12), 1371-1375.
- [11]. Suk, H. I., Sin, B. K., & Lee, S. W. (2010). Hand gesture recognition based on dynamic Bayesian network framework. Pattern recognition, 43(9), 3059- 3072.
- [12]. Admasu, Y. F., & Raimond, K. (2010, November). Ethiopian sign language recognition using Artificial Neural Network. In 2010 10th International Conference on Intelligent Systems Design and Applications (pp. 995-1000). IEEE.
- [13]. Pigou, L., Dieleman, S., Kindermans, P. J., & Schrauwen, B. (2014, September). Sign language recognition using convolutional neural networks. In European Conference on Computer Vision (pp. 572-578). Springer, Cham.
- [14]. Escalera, S., Baró, X., Gonzalez, J., Bautista, M. A., Madadi, M., Reyes, M., ... & Guyon, I. (2014, September). Chalearn looking at people challenge 2014: Dataset and results. In European Conference on Computer Vision (pp. 459-473). Springer, Cham.
- [15]. <https://github.com/Rishi-Sanmitra/RealTimeSignLanguageDetection> - Project location on GitHub.

## 9.2 WEBSITES

- [www.w3schools.com](http://www.w3schools.com)
- [www.towardsdatascience.com](http://www.towardsdatascience.com)
- [www.stackoverflow.com](http://www.stackoverflow.com)
- [www.wikipedia.com](http://www.wikipedia.com)
- [www.pyimagesearch.com](http://www.pyimagesearch.com)



# IJRESM

International Journal of Research in  
Engineering, Science and Management

[www.ijresm.com](http://www.ijresm.com) [support@ijresm.com](mailto:support@ijresm.com)

SJIF Impact Factor: 4.308

## Certificate

It is here by certified that the manuscript entitled

**Machine Learning Based Real Time Sign Language Detection**

by

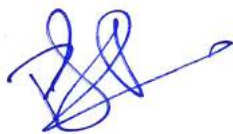
**P. Rishi Sanmitra**

has been published in

Volume 4, Issue 6, June 2021

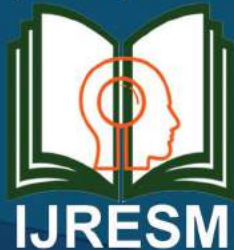
in

International Journal of Research in  
Engineering, Science and Management



Editor-in-Chief (IJRESM)

All the best for your future endeavors



# IJRESM

International Journal of Research in  
Engineering, Science and Management

www.ijresm.com support@ijresm.com

SJIF Impact Factor: 4.308

## Certificate

It is here by certified that the manuscript entitled

**Machine Learning Based Real Time Sign Language Detection**

by

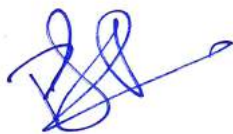
**V. V. Sai Sowmya**

has been published in

Volume 4, Issue 6, June 2021

in

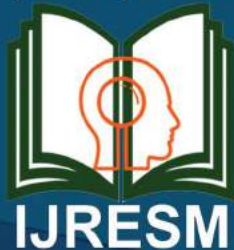
International Journal of Research in  
Engineering, Science and Management



Editor-in-Chief (IJRESM)

All the best for your future endeavors





# IJRESM

International Journal of Research in  
Engineering, Science and Management

[www.ijresm.com](http://www.ijresm.com) [support@ijresm.com](mailto:support@ijresm.com)

SJIF Impact Factor: 4.308

## Certificate

It is here by certified that the manuscript entitled

**Machine Learning Based Real Time Sign Language Detection**

by

**K. Lalithanjana**

has been published in

Volume 4, Issue 6, June 2021

in

International Journal of Research in  
Engineering, Science and Management



Editor-in-Chief (IJRESM)

All the best for your future endeavors

# Machine Learning Based Real Time Sign Language Detection

P. Rishi Sanmitra<sup>1</sup>, V. V. Sai Sowmya<sup>2</sup>, K. Lalithanjana<sup>3\*</sup>

<sup>1,2,3</sup>Student, Department of Computer Science and Engineering, CMR Technical Campus, Hyderabad, India

**Abstract:** In this paper, a real time ML based system was built for the Sign Language Detection using images that have been captured with the help of a PC camera. The main purpose of this project is to design a system for the differently abled people to communicate with others with ease. This model is one of the first models to detect signs irrespective of their sign language standards (i.e., the American Standard or the Indian Standard). The existing digital translators are very slow since every alphabet has to be gestured out and the amount of time it would take to just form a simple sentence would be a lot. This model, which was trained using the SSD ML Algorithm, overcomes the above problem by directly recognizing the signs as words instead of alphabets. This model was trained using a set of 20 images for a particular sign in different conditions such as different lighting, different skin tones, backgrounds, etc., in order to increase the accuracy of detecting the gesture. The system displayed a high accuracy for all the datasets when new test data, which had not been used in the training, were introduced. The results have shown a high accuracy of 85% for the sign detection.

**Keywords:** Deep Learning SSD ML algorithm, LabelImg software, real time, TensorFlow object detection module.

## 1. Introduction

A very few people know how to communicate using sign language as it is not a mandatory language to learn. This makes it difficult for differently abled people to communicate with others.

The most common means to communicate with them is with the help of human interpreters, which is again very expensive and not many can afford it. There are many different sign languages in the world. There are around 200 sign languages in the world including Chinese, Spanish, Irish, American Sign Language and Indian Sign Language, which are the most commonly used sign languages.

The ML based Sign Language Detection system aims at communicating with differently abled people without the help of any expensive human interpreter. This model translates the signs/gestures captured into text so that the user can simply read and know what the person is trying to convey irrespective of whether the user has knowledge about the sign language or not.

## 2. Literature Survey

The first approach in relation to sign language recognition was by Bergh in 2011 [1]. Haar wavelets and database

searching were employed to build a hand gesture recognition system. Although this system gives good results, it only considers six classes of gestures.

In a study by Balbin et al. [2], the system only recognized five Filipino words and used colored gloves for hand position recognition; our model can be trained for different gestures and can be recognized without any colored gloves and using only bare hands.

In our model the images are captured using a PC Cam and were able to get an accuracy of 75% at an average. In other models these were captured using motion sensors, such as electromyography (EMG) sensors [3], RGB cameras [4], Kinect sensors [5] and their combinations. Although the accuracy of detecting the signs is high, they also have limitations; first is their cost, as they require large-size datasets with diverse sign motion they go to a high-end computer with powerful specifications; whereas in our model this can be achieved with minimum specifications.

The SSD model was also adopted for hand detection. The proposed model was evaluated based on the IsoGD dataset, which achieved 4.25% accuracy [6].

In 2016, with the aim of real-time object detection in testing images, two novel algorithms came out, namely, YOLO and SSD [7], [8]. YOLO uses CNN to reduce the spatial dimension detection box. It performs a linear regression to make boundary box predictions. In the case of SSD, the size of the detecting box is usually fixed and used for simultaneous size detection. Therefore, the purported advantage of SSD is known to be the simultaneous detection of objects with various sizes.

In comparison to other systems which only recognized ASL alphabets, our model is mainly for recognizing gestures, making it more useful and effective. In the literature [9]-[12], the systems only recognized ASL alphabets.

## 3. Methodology

### A. System Architecture

In this project, a real time sign recognition ML model was built with the help of LabelImg software and TensorFlow Object Detection API, using real coloring images. This system was divided into three main phases; Initially we wrote some code to automate the picture taking process, once the pictures were taken, we used the LabelImg software to segregate these

\*Corresponding author: [lalithanjanakollipara@gmail.com](mailto:lalithanjanakollipara@gmail.com)

images into the appropriate labels. These labels are named in such a way that they express the meaning of the gesture made. Once the labelling of the images was done, we have two sets of files for each image taken, one which has the actual image in it and the other being an XML file which contains information of where the model should be looking in the image during the training process. Once these files are generated, the training process begins, where the Machine is going to use a Deep Learning SSD ML algorithm to extract features from the desired image. Finally, after the model has been trained, it allows for the Sign Language Detection part to begin. To achieve the detection, we are using the TensorFlow Object Detection API where the extracted features from the images taken are passed onto the TensorFlow module which is going to make comparisons with the real time video present in the frame. On detection of any of these features it is going to generate a bounding box around the gesture and make the prediction. The prediction is going to be the same as the label of the image, hence it is very important to understand the gesture made so as to name the label correctly, a wrongly named label could result in a wrong prediction.

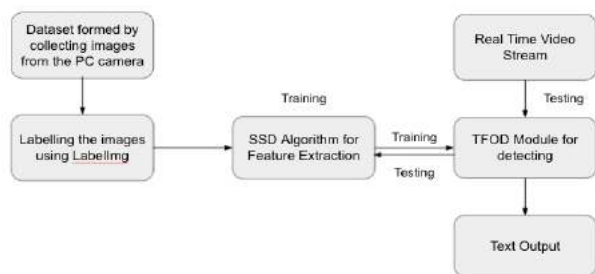


Fig. 1. System architecture

**B. Dataset Creation**

The LabelImg software is used for graphically labelling the images that is further used when recognizing the images. We have to keep in mind that labelling has to be done correctly i.e., the gesture should be labelled with a right label so that we get the gestures recognized correctly later with the right label. Once the images are labelled and saved an XML file is created for that image. This XML file contains the information about where the model should be looking in the image during the training process.

This model is trained for 5 different gestures hence 5 different labels were used for labelling them. For each gesture 20 images were used that are clicked in different angles. A code is used to take the images automatically and save them in a particular folder.

The labelling is done by drawing a box around the gesture made. This box is called the Ground Truth which means a set of measurements that is known to be much more accurate than measurements from the system you are testing. The below figure (i.e., Fig. 2) demonstrates how the images are labelled using LabelImg software.

The XML file associated with a labelled image showing where the model has to search for the gesture while training the ML model is shown in the below figure (i.e., Fig. 3).

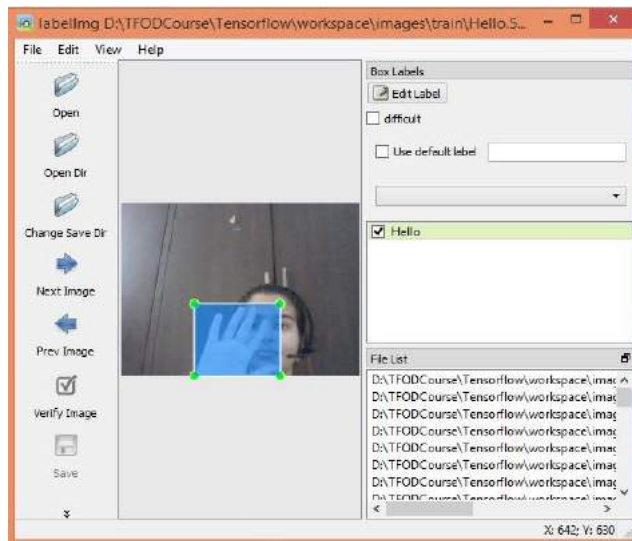


Fig. 2. Labelling the gestures using LabelImg

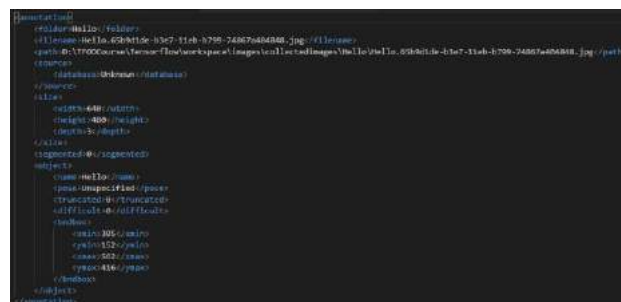


Fig. 3. XML file of a labelled image

**C. Training and Testing**

Out of the 20 images collected along with the generated XML files for each image, 5 were used for testing and the remaining 15 were used for training the model. The ML model was trained using the Deep Learning SSD ML Algorithm and tested using the TensorFlow Object Detection API.

SSD (Single Shot Detection) algorithm is designed for object detection in real-time. Faster R-CNN uses a region proposal network to create boundary boxes and utilizes those boxes to classify objects. While it is considered state-of-the-art in accuracy, the whole process runs at 7 frames per second. Far below what real-time processing needs. SSD speeds up the process by eliminating the need for the region proposal network. To recover the drop in accuracy, SSD applies a few improvements including multi-scale features and default boxes. These improvements allow SSD to match the Faster R-CNN’s accuracy using lower resolution images, which further pushes the speed higher.

The SSD architecture is a single convolution network that learns to predict bounding box locations and classify these locations in one pass. Hence, SSD can be trained end-to-end. The SSD network consists of base architecture (MobileNet in this case) followed by several convolution layers.

TensorFlow is an open-source library for numerical computation and large-scale machine learning that eases Google Brain TensorFlow, the process of acquiring data, training models, serving predictions, and refining future results.



The TensorFlow Object Detection API is an open-source framework built on top of TensorFlow that makes it easy to construct, train and deploy object detection models. There are already pre-trained models in their framework which are referred to as Model Zoo. It includes a collection of pre-trained models trained on various datasets such as the COCO (Common Objects in Context) dataset, the KITTI dataset, and the Open Images Dataset. The TensorFlow object detection API is the framework for creating a deep learning network that solves object detection problems.

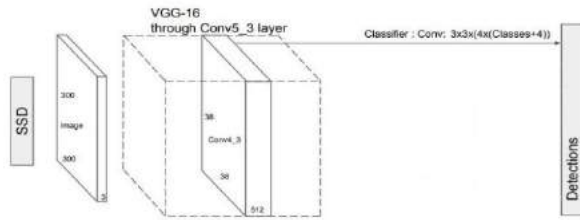


Fig. 4. SSD network architecture

TensorFlow bundles together Machine Learning and Deep Learning models and algorithms. It uses Python as a convenient front-end and runs it efficiently in optimized C++. TensorFlow allows developers to create a graph of computations to perform. Each node in the graph represents a mathematical operation and each connection represents data. Hence, instead of dealing with low-details like figuring out proper ways to hitch the output of one function to the input of another, the developer can focus on the overall logic of the application.

We use ‘Checkpoints’ that are save points which a model generates to keep track of how much it has trained itself. In case the training process is interrupted, it would simply start itself again from the checkpoint. Since the training process can be very time consuming, this mechanism allows the model to save itself from system failures. The learning rate of our model when used 10000 steps for training is shown below in Fig. 5.

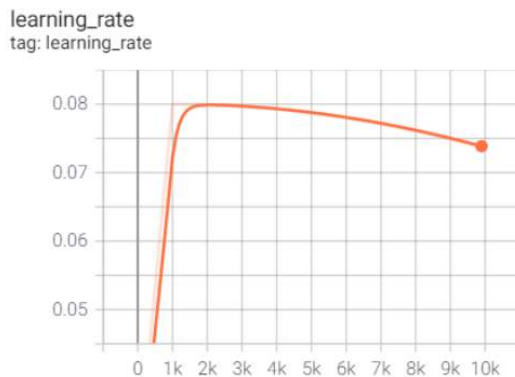


Fig. 5. Learning rate of 10000 steps training model

A loss function is used to optimize the machine learning algorithm. The loss is calculated on training and testing, and its interpretation is based on how well the model is doing in these two sets. It is the sum of errors made for each example in training or testing sets. Loss value implies how poorly or well a

model behaves after each iteration of optimization. The loss at each iteration of our machine learning model has been decreasing which indicates a better accuracy of model for detection. The loss of our model is shown in the below Fig. 6.

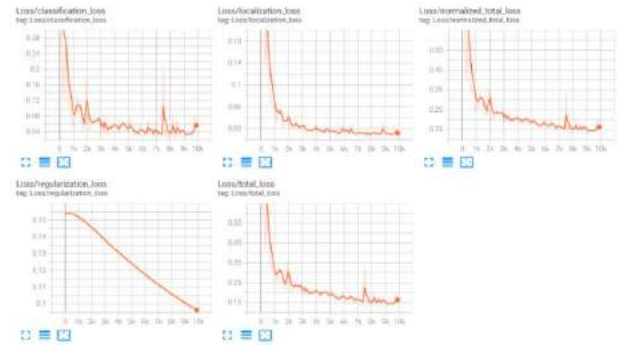


Fig. 6. Loss of the Machine Learning model

The localization loss is the mismatch between the ground truth box and the predicted boundary box. SSD only penalizes predictions from positive matches. Only the predictions from the positive matches to get closer to the ground truth is required. Negative matches can be ignored. Ground truth box is the box that is created in the LabelImg software while creating the labels and the predicted boundary box is the box that is predicted by the model while testing the images. The localization loss for our model is 0.05 as shown in Fig. 9.

The localization loss between the predicted box  $l$  and the ground truth box  $g$  is defined as the smooth L1 loss with  $cx, cy$  as the offset to the default bounding box  $d$  of width  $w$  and height  $h$ .

$$L_{loc}(x, l, g) = \sum_{i \in Pos} \sum_{m \in \{cx, cy, w, h\}} x_{ij}^{smooth_{L1}}(|l_i^m - g_j^m|)$$

$$g_j^x = (g_j^x - d_j^x) / d_j^x \quad g_j^y = (g_j^y - d_j^y) / d_j^y$$

$$\hat{g}_j^x = \log\left(\frac{g_j^x}{d_j^x}\right) \quad \hat{g}_j^y = \log\left(\frac{g_j^y}{d_j^y}\right)$$

$$x_{ij}^p = \begin{cases} 1 & \text{if IoU} > 0.5 \text{ between default box } i \text{ and ground true box } j \text{ on class } p \\ 0 & \text{otherwise} \end{cases}$$

Fig. 7. Formula for calculating localization loss

The confidence loss is the loss of making a class prediction. For every positive match prediction, the loss is penalized according to the confidence score of the corresponding class. For negative match predictions, the loss is penalized according to the confidence score of the class “0”: class “0” classifies no object is detected. The confidence loss for our model is 0.19 as shown in Fig. 9.

It is calculated as the softmax loss over multiple classes confidences  $c$  (class score).

$$L_{conf}(x, c) = - \sum_{i \in Pos} x_{ij}^p \log(\hat{c}_i^p) - \sum_{i \in Neg} \log(\hat{c}_i^0) \quad \text{where} \quad \hat{c}_i^p = \frac{\exp(c_i^p)}{\sum_p \exp(c_i^p)}$$

where  $N$  is the number of matched default boxes.

Fig. 8. Formula for calculating confidence loss

The below image (i.e., Fig. 9.) represents the evaluation results and evaluation metrics for a 10000-step machine learning model. An evaluation metric consists of the average precision and average recall. For each precision and recall an IOU is calculated. IOU stands for Intersection Over Union

which determines the ratio of area of intersection between the Ground Truth and predicted box to the area of union between the Ground Truth and Predicted box (as shown in Fig. 10)

```

C:\Users\Pranav> python Tensorflow\model\train.py --input_dir=train_data --output_dir=model --ckpt=model_ckpt --model_dir=Tensorflow\object_detection\ --model_name=ssd_300 --num_epochs=100
Accumulating evaluation results...
Done (14.04s)
Average Precision (AP) @ IoU=0.50 (0.50) area= all mAP=0.895
Average Precision (AP) @ IoU=0.50 area= all mAP=1.000
Average Precision (AP) @ IoU=0.75 area= all mAP=1.000
Average Precision (AP) @ IoU=0.50 area= small mAP=1.000
Average Precision (AP) @ IoU=0.50 area= medium mAP=1.000
Average Precision (AP) @ IoU=0.50 area= large mAP=0.993
Average Recall (AR) @ IoU=0.50 area= all mAP=1.000
Average Recall (AR) @ IoU=0.50 area= small mAP=1.000
Average Recall (AR) @ IoU=0.50 area= medium mAP=1.000
Average Recall (AR) @ IoU=0.50 area= large mAP=0.995
INFO:tensorflow:Final metrics at step 10000
0515 17:00:48.848389 30002 model_lib_v2.py:992] Final metrics at step 10000
INFO:tensorflow: + DetectionBoxes_Precision/mAP: 0.895446
0515 17:00:48.805177 30002 model_lib_v2.py:992] + DetectionBoxes_Precision/mAP: 0.095446
INFO:tensorflow: + DetectionBoxes_Precision/mAP_small: 1.000000
0515 17:00:48.809134 30002 model_lib_v2.py:992] + DetectionBoxes_Precision/mAP_medium: 1.000000
INFO:tensorflow: + DetectionBoxes_Precision/mAP_large: 0.992999
0515 17:00:48.805117 30002 model_lib_v2.py:992] + DetectionBoxes_Precision/mAP_small: 1.000000
INFO:tensorflow: + DetectionBoxes_Precision/mAP_medium: 1.000000
0515 17:00:48.809187 30002 model_lib_v2.py:992] + DetectionBoxes_Precision/mAP_large: 0.995000
INFO:tensorflow: + DetectionBoxes_Precision/mAP (large): 0.995000
0515 17:00:48.821182 30002 model_lib_v2.py:992] + DetectionBoxes_Precision/mAP (large): 0.995000
INFO:tensorflow: + DetectionBoxes_Recall/mAP: 0.905000
0515 17:00:48.822841 30002 model_lib_v2.py:992] + DetectionBoxes_Recall/mAP: 0.905000
INFO:tensorflow: + DetectionBoxes_Recall/mAP_small: 0.905000
0515 17:00:48.825981 30002 model_lib_v2.py:992] + DetectionBoxes_Recall/mAP_medium: 0.905000
INFO:tensorflow: + DetectionBoxes_Recall/mAP_large: 0.905000
0515 17:00:48.827887 30002 model_lib_v2.py:992] + DetectionBoxes_Recall/mAP_small: 1.000000
0515 17:00:48.830953 30002 model_lib_v2.py:992] + DetectionBoxes_Recall/mAP_medium: 1.000000
0515 17:00:48.831713 30002 model_lib_v2.py:992] + DetectionBoxes_Recall/mAP_large: 0.905000
INFO:tensorflow: + Loss/Localization_loss: 0.021289
0515 17:00:48.808241 30002 model_lib_v2.py:992] + Loss/Localization_loss: 0.021289
INFO:tensorflow: + Loss/classification_loss: 0.844222
0515 17:00:48.808256 30002 model_lib_v2.py:992] + Loss/classification_loss: 0.844222
INFO:tensorflow: + Loss/regularization_loss: 0.001017
0515 17:00:48.808256 30002 model_lib_v2.py:992] + Loss/regularization_loss: 0.001017
INFO:tensorflow: + Loss/total_loss: 0.286209
0515 17:00:48.808308 30002 model_lib_v2.py:992] + Loss/total_loss: 0.286209
    
```

Fig. 9. Evaluation results and evaluation metrics

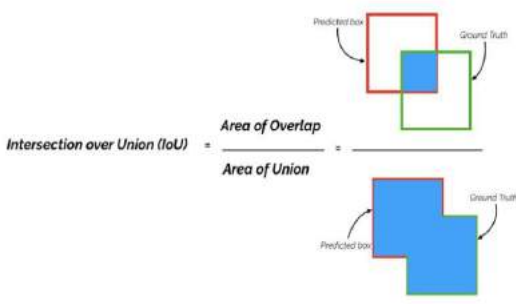


Fig. 10. Intersection over Union (IoU)

```

C:\Users\Pranav>
INFO:tensorflow:Step 7700 per-step time 0.174s loss=0.161
0515 16:54:50.788918 16348 model_lib_v2.py:683] Step 7700 per-step time 0.174s loss=0.161
INFO:tensorflow:Step 7800 per-step time 0.173s loss=0.137
0515 16:55:08.102036 16348 model_lib_v2.py:683] Step 7800 per-step time 0.173s loss=0.137
0515 16:55:08.102036 16348 model_lib_v2.py:683] Step 7900 per-step time 0.174s loss=0.197
0515 16:55:25.463247 16348 model_lib_v2.py:683] Step 7900 per-step time 0.174s loss=0.197
INFO:tensorflow:Step 8000 per-step time 0.173s loss=0.166
0515 16:55:42.735772 16348 model_lib_v2.py:683] Step 8000 per-step time 0.173s loss=0.166
INFO:tensorflow:Step 8100 per-step time 0.181s loss=0.152
0515 16:56:00.809610 16348 model_lib_v2.py:683] Step 8100 per-step time 0.181s loss=0.152
INFO:tensorflow:Step 8200 per-step time 0.177s loss=0.144
0515 16:56:18.493315 16348 model_lib_v2.py:683] Step 8200 per-step time 0.177s loss=0.144
0515 16:56:36.111662 16348 model_lib_v2.py:683] Step 8300 per-step time 0.176s loss=0.136
0515 16:56:54.092024 16348 model_lib_v2.py:683] Step 8300 per-step time 0.176s loss=0.154
0515 16:56:54.092024 16348 model_lib_v2.py:683] Step 8400 per-step time 0.179s loss=0.154
INFO:tensorflow:Step 8500 per-step time 0.180s loss=0.165
0515 16:57:12.006381 16348 model_lib_v2.py:683] Step 8500 per-step time 0.180s loss=0.165
INFO:tensorflow:Step 8600 per-step time 0.179s loss=0.146
0515 16:57:29.890624 16348 model_lib_v2.py:683] Step 8600 per-step time 0.179s loss=0.146
0515 16:57:47.810624 16348 model_lib_v2.py:683] Step 8700 per-step time 0.180s loss=0.134
0515 16:58:05.729270 16348 model_lib_v2.py:683] Step 8700 per-step time 0.180s loss=0.134
INFO:tensorflow:Step 8800 per-step time 0.181s loss=0.144
0515 16:58:23.648024 16348 model_lib_v2.py:683] Step 8800 per-step time 0.181s loss=0.144
0515 16:58:41.567024 16348 model_lib_v2.py:683] Step 8900 per-step time 0.182s loss=0.155
0515 16:58:59.486270 16348 model_lib_v2.py:683] Step 8900 per-step time 0.182s loss=0.155
INFO:tensorflow:Step 9000 per-step time 0.180s loss=0.176
0515 16:59:17.405270 16348 model_lib_v2.py:683] Step 9000 per-step time 0.180s loss=0.176
0515 16:59:35.324270 16348 model_lib_v2.py:683] Step 9100 per-step time 0.180s loss=0.181
0515 16:59:53.243270 16348 model_lib_v2.py:683] Step 9100 per-step time 0.180s loss=0.181
INFO:tensorflow:Step 9200 per-step time 0.180s loss=0.180
0515 16:59:53.243270 16348 model_lib_v2.py:683] Step 9200 per-step time 0.180s loss=0.180
0515 16:59:53.243270 16348 model_lib_v2.py:683] Step 9300 per-step time 0.180s loss=0.143
0515 16:59:53.243270 16348 model_lib_v2.py:683] Step 9300 per-step time 0.180s loss=0.143
0515 16:59:53.243270 16348 model_lib_v2.py:683] Step 9400 per-step time 0.178s loss=0.133
0515 16:59:53.243270 16348 model_lib_v2.py:683] Step 9400 per-step time 0.178s loss=0.133
0515 16:59:53.243270 16348 model_lib_v2.py:683] Step 9500 per-step time 0.179s loss=0.137
0515 16:59:53.243270 16348 model_lib_v2.py:683] Step 9500 per-step time 0.179s loss=0.137
0515 16:59:53.243270 16348 model_lib_v2.py:683] Step 9600 per-step time 0.179s loss=0.137
0515 16:59:53.243270 16348 model_lib_v2.py:683] Step 9600 per-step time 0.179s loss=0.137
0515 16:59:53.243270 16348 model_lib_v2.py:683] Step 9700 per-step time 0.180s loss=0.133
0515 16:59:53.243270 16348 model_lib_v2.py:683] Step 9700 per-step time 0.180s loss=0.133
0515 16:59:53.243270 16348 model_lib_v2.py:683] Step 9800 per-step time 0.180s loss=0.140
0515 16:59:53.243270 16348 model_lib_v2.py:683] Step 9800 per-step time 0.180s loss=0.140
0515 16:59:53.243270 16348 model_lib_v2.py:683] Step 9900 per-step time 0.179s loss=0.137
0515 16:59:53.243270 16348 model_lib_v2.py:683] Step 9900 per-step time 0.179s loss=0.137
0515 16:59:53.243270 16348 model_lib_v2.py:683] Step 10000 per-step time 0.179s loss=0.135
0515 16:59:53.243270 16348 model_lib_v2.py:683] Step 10000 per-step time 0.179s loss=0.135
    
```

Fig. 11. Loss at each iteration

A loss function is a mathematical formula used to produce loss values during training time. During training, the performance of a model is measured by the loss (L) that the model produces for each sample or batch of samples. The loss essentially measures how “far” the predicted values (y) are from the expected value (y). If y is far away (very different) from y, then the loss will be high. However, if y is close to y then the loss is low. The model uses the loss as an “indicator” to update its parameters so that it can produce very small losses in future predictions. That means producing y that are very close to y.

The figure (i.e., Fig. 11.) shows the loss incurred at each step while training the model. The lowest loss recorded is 0.133 at step 9400 and at step 9700.

#### 4. Results and Discussions

A real-time Sign Language Detection with a SSD algorithm using real coloring images from a PC camera was introduced. In this paper, signs are translated into text statements to help the differently abled people to communicate with others with ease. This system showed good results by taking advantage of deep learning techniques. This section discusses the results obtained by the system.

Fig. 12. shows an accuracy of 90% for the recognition of the sign ‘No’ by the system.



Fig. 12. Gesture recognition for No

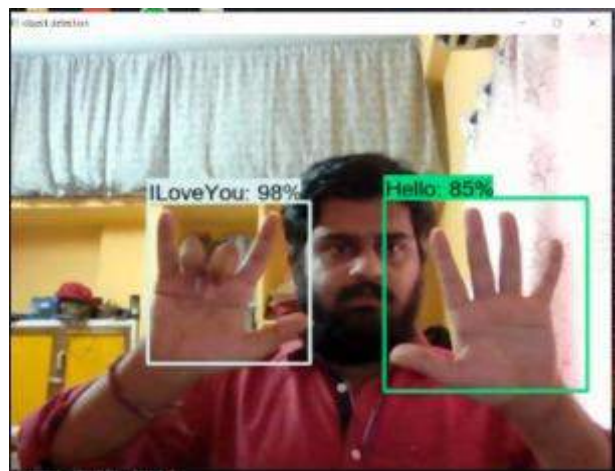


Fig. 13. Gesture recognition for I Love You and Hello

Two gestures can also be recognized simultaneously using our system. The accuracy has shown to be the same irrespective of two gestures being made simultaneously. In Fig. 13. we can see that both the gestures are being recognized without any difficulty.

Apart from the gestures recognized above, there are two more gestures that we used for training the model. Total five gestures were used to train the machine learning model by taking 20 images for each model in different angles, backgrounds, skin tones, lighting and other various situations. Out of the 20 images collected, 15 were used for training and 5 were for testing. All the images were converted into gray scale images as shown in Fig. 14., for training the ML model. The results have shown up to an average accuracy of 85%.



Fig. 14. Images for training in grayscale

## 5. Conclusion

In this paper, a real-time ML based Sign Language Recognition system was built using real coloring images that were taken with the help of a PC camera. New datasets were built to contain a wider variety of features for example different lightings, different skin tones, different backgrounds, and a wide variety of hand gestures. The system achieved a maximal accuracy of about 75% for training and 85% for the validation set. In addition, the system showed a high accuracy with the introduction of new test data that had not been used in the training. There is a lot of scope for this project, since we have the ability to label these images, we can label the gestures in

any language required, allowing the user to communicate with others irrespective of the language boundaries.

## References

- [1] M. Van den Bergh and L. Van Gool, "Combining RGB and ToF cameras for real-time 3D hand gesture interaction," *2011 IEEE Workshop on Applications of Computer Vision (WACV)*, 2011, pp. 66-72.
- [2] J. R. Balbin et al., "Sign language word translator using Neural Networks for the Aurally Impaired as a tool for communication," *2016 6th IEEE International Conference on Control System, Computing and Engineering (ICCSCE)*, 2016, pp. 425-429.
- [3] J. Wu, Z. Tian, L. Sun, L. Estevez and R. Jafari, "Real-time American Sign Language Recognition using wrist-worn motion and surface EMG sensors," *2015 IEEE 12th International Conference on Wearable and Implantable Body Sensor Networks (BSN)*, 2015, pp. 1-6.
- [4] D. Mart, Sign Language Translator Using Microsoft Kinect XBOX 360 TM, 2012, pp. 1-76.
- [5] Cao Dong, M. C. Leu and Z. Yin, "American Sign Language alphabet recognition using Microsoft Kinect," *2015 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2015, pp. 44-52.
- [6] Rastgoo R, Kiani K, Escalera S, "Video-based isolated hand sign language recognition using a deep cascaded model," in *Multimed. Tools Appl.* 2020, pp. 22965–22987.
- [7] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 779-788.
- [8] Liu W, Anguelov D, Erhan D, et al., "SSD: Single shot multibox detector," in *Proceedings of the European Conference on Computer Vision*, New York: Springer; 2016, pp. 21–37.
- [9] T. Kim, G. Shakhnarovich, and K. Livescu, "Finger-spelling recognition with semi-markov conditional random fields," in *Proc. 2013 IEEE International Conference on Computer Vision*, 2013, pp. 1521–1528.
- [10] N. Pugeault and R. Bowden, "Spelling it out: Real-time asl finger-spelling recognition," in *Proc. 2011 IEEE International Conference on Computer Vision Workshop*, 2011, pp. 1114–1119
- [11] S. Shahriar, A. Siddiquee, T. Islam, A. Ghosh, R. Chakraborty, A. I. Khan, C. Shahnaz, and S. A. Fattah, "Real-time american sign language recognition using skin segmentation and image category classification with convolutional neural network and deep learning," in *Proc. TENCON 2018-2018 IEEE Region 10 Conference*, 2018, pp. 1168-1171.
- [12] R. Daroya, D. Peralta, and P. Naval, "Alphabet sign language image classification using deep learning," in *Proc. TENCON 2018-2018 IEEE Region 10 Conference*, 2018, pp. 646-650.