SRI RAMACHANDRA
INSTITUTE OF HIGHER EDUCATION AND RESEARCH
(Category - I Deemed to be University) Porur, Chennai
SRI RAMACHANDRA ENGINEERING AND TECHNOLOGY

DEVOPS AND UNIT-TESTING

INTERNSHIP REPORT

Quarter IV (Year 1)

Submitted by

S Vishvajith Reddy E0220018

In partial fulfilment for the award of the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

(Cyber Security & Internet of Things)

Sri Ramachandra Engineering and Technology

Sri Ramachandra Institute of Higher Education and Research, Porur, Chennai -600116

JULY, 2020

SRI RAMACHANDRA
INSTITUTE OF HIGHER EDUCATION AND RESEARCH
(Category - I Deemed to be University) Porur, Chennai
SRI RAMACHANDRA ENGINEERING AND TECHNOLOGY

DEVOPS AND UNIT-TESTING

INTERNSHIP REPORT

Quarter IV (Year 1)

Submitted by

S Vishvajith Reddy E0220018

In partial fulfilment for the award of the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

(Cyber Security & Internet Of Things)

Sri Ramachandra Engineering and Technology

Sri Ramachandra Institute of Higher Education and Research, Porur, Chennai -600116

JULY, 2020

SRI RAMACHANDRA
INSTITUTE OF HIGHER EDUCATION AND RESEARCH
(Category - I Deemed to be University) Porur, Chennai
SRI RAMACHANDRA ENGINEERING AND TECHNOLOGY

BONAFIDE CERTIFICATE

Certified that this project report **"Devops and Unit-Testing"** is the bonafide work of **S Vishvajith Reddy Reg No. E0220018** who carried out the internship work under my supervision.

Signature of Faculty Mentor

Chiranjeevi N

Assistant Professor

Sri Ramachandra Engineering and Technology

Porur

Chennai-600116

Signature of Vice-Principal

Prof. M. Prema

Vice-Principal

Sri Ramachandra Engineering and Technology

Porur

Chennai-600116

Evaluation Date:

Table of Contents

ACKNOWLEDGEMENT

I express my sincere gratitude to our Chancellor, Vice-Chancellor and our sincere gratitude to our Provost **Dr. Raju** and our Vice-Principal **Prof. Prema** for their support and for providing the required facilities for carrying out this study.

I wish to thank my faculty mentor, **Prof. Chiranjeevi N,** Department of Computer Science and Engineering, Sri Ramachandra Engineering and Technology for extending help and encouragement throughout the project. Without his/her continuous guidance and persistent help, this project would not have been a success for me.

I am grateful to Department of Computer Science and Engineering, Sri Ramachandra Engineering and Technology, our beloved parents and friends for extending the support, who helped us to overcome obstacles in the study.

1.DOMAIN INTRODUCTION

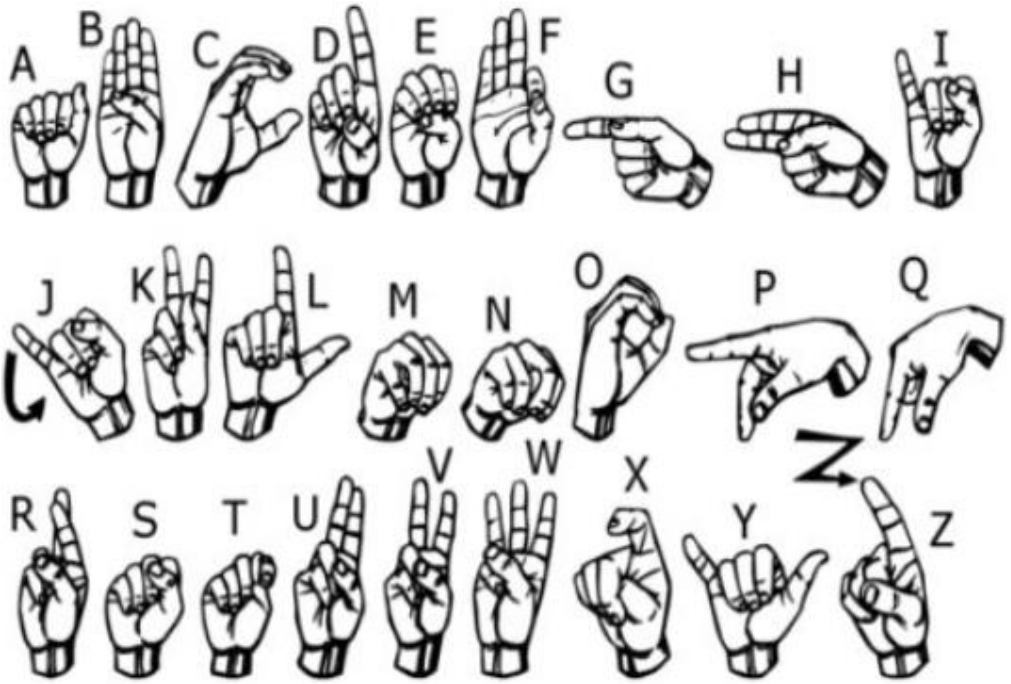
Data Analytics:

Analytics is the systematic computational analysis of data or statistics. It is used for the discovery, interpretation, and communication of meaningful patterns in data. It also entails applying data patterns towards effective decision-making. It can be valuable in areas rich with recorded information; analytics relies on the simultaneous application of statistics, computer programming and operations research to quantify performance.

Organizations may apply analytics to business data to describe, predict, and improve business performance. Specifically, areas within analytics include predictive analytics, prescriptive analytics, enterprise decision management, descriptive analytics, cognitive analytics, Big Data Analytics, retail analytics, supply chain analytics, store assortment and stock-keeping unit optimization, marketing optimization and marketing mix modelling, web analytics, call analytics, speech analytics, sales force sizing and optimization, price and promotion modelling, predictive science, graph analytics, credit risk analysis, and fraud analytics. Since analytics can require extensive computation (see big data), the algorithms and software used for analytics harness the most current methods in computer science, statistics, and mathematics

2.Objective:

Sign language is one of the oldest and most natural form of language for communication, but since most people do not know sign language and interpreters are very difficult to come by we have come up with a real time method using neural networks for fingerspelling based american sign language. Our method provides 90.5% accuracy for the 26 letters of the alphabet.



3. Technology used:

3.1 TensorFlow:

TensorFlow is a free and open-source software library for machine learning. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks. TensorFlow is a symbolic math library based on dataflow and differentiable programming.

3.2 OpenCV:

OpenCV is a library of programming functions mainly aimed at real-time computer vision. Originally developed by Intel, it was later supported by Willow Garage then Itrez. The library is cross-platform and free for use under the open-source Apache 2 License.

3.3 Mediapipe:

Mediapipe is a cross-platform framework for building multimodal applied machine learning pipelines. ... video, audio, any time series data), cross platform (i.e. Android, iOS, web, edge devices) applied ML pipelines.

3.4 SKLearn:

Scikit-learn is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines,

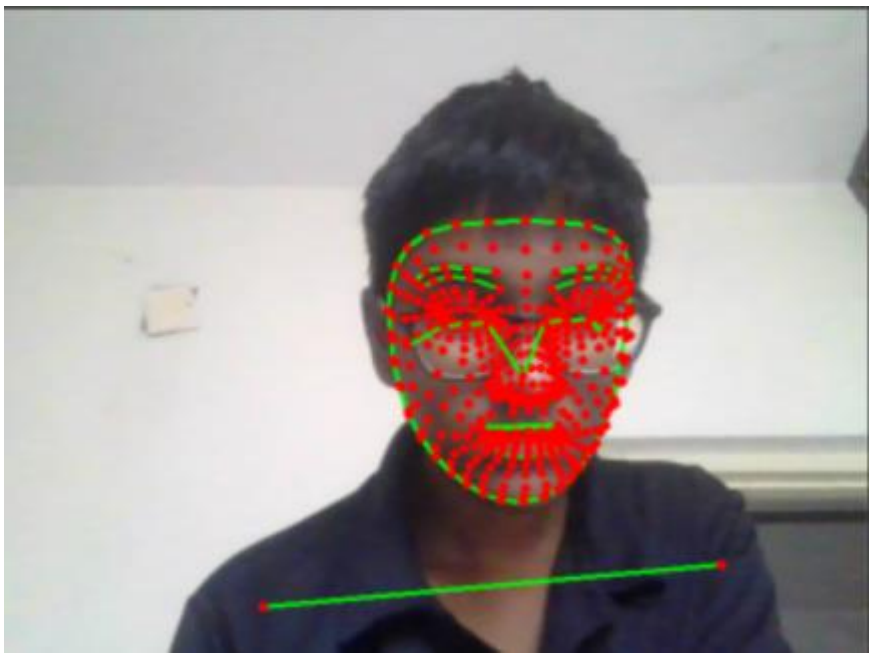
3.5 Matplotlin:

Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK.

3.6 Jupyter Notebook:

Project Jupyter is a project and community whose goal is to "develop open-source software, open-standards, and services for interactive computing across dozens of programming languages". It was spun off from I Python in 2014 by Fernando Pérez.

4.Sample Output:



5 Code:

1. Import and Install Dependencies

!pip install tensorflow==2.4.1 tensorflow-gpu==2.4.1 opencv-python mediapipe sklearn matplotlib

```
import cv2
import numpy as np
import os
from matplotlib import pyplot as plt
import time
import mediapipe as mp
```

2. Key points using MP Holistic

```
mp_holistic = mp.solutions.holistic # Holistic model

mp_drawing = mp.solutions.drawing_utils # Drawing utilities

def mediapipe_detection(image, model):

    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB) # COLOR CONVERSION BGR 2 RGB
```

```
    image.flags.writeable = False                  # Image is no longer writeable
    results = model.process(image)                 # Make prediction
    image.flags.writeable = True                   # Image is now writeable
    image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR) # COLOR COVERSION RGB 2 BGR
    return image, results

def draw_landmarks(image, results):
    mp_drawing.draw_landmarks(image, results.face_landmarks, mp_holistic.FACE_CONNECTIONS) # Draw face connections
    mp_drawing.draw_landmarks(image, results.pose_landmarks, mp_holistic.POSE_CONNECTIONS) # Draw pose connections
    mp_drawing.draw_landmarks(image, results.left_hand_landmarks, mp_holistic.HAND_CONNECTIONS) # Draw left hand connections
    mp_drawing.draw_landmarks(image, results.right_hand_landmarks, mp_holistic.HAND_CONNECTIONS) # Draw right hand connections

def draw_styled_landmarks(image, results):
    # Draw face connections
    mp_drawing.draw_landmarks(image, results.face_landmarks, mp_holistic.FACE_CONNECTIONS,
                              mp_drawing.DrawingSpec(color=(80,110,10), thickness=1, circle_radius=1),
                              mp_drawing.DrawingSpec(color=(80,256,121), thickness=1, circle_radius=1)
                              )
    # Draw pose connections
    mp_drawing.draw_landmarks(image, results.pose_landmarks, mp_holistic.POSE_CONNECTIONS,
                              mp_drawing.DrawingSpec(color=(80,22,10), thickness=2, circle_radius=4),
                              mp_drawing.DrawingSpec(color=(80,44,121), thickness=2, circle_radius=2)
```

```
                    )
    # Draw left hand connections
    mp_drawing.draw_landmarks(image, results.left_hand_landmarks,
mp_holistic.HAND_CONNECTIONS,
                    mp_drawing.DrawingSpec(color=(121,22,76), thickness=2,
circle_radius=4),
                    mp_drawing.DrawingSpec(color=(121,44,250), thickness=2,
circle_radius=2)
                    )
    # Draw right hand connections
    mp_drawing.draw_landmarks(image, results.right_hand_landmarks,
mp_holistic.HAND_CONNECTIONS,
                    mp_drawing.DrawingSpec(color=(245,117,66), thickness=2,
circle_radius=4),
                    mp_drawing.DrawingSpec(color=(245,66,230), thickness=2,
circle_radius=2)
                    )
cap = cv2.VideoCapture(0)
# Set mediapipe model
with mp_holistic.Holistic(min_detection_confidence=0.5,
min_tracking_confidence=0.5) as holistic:
    while cap.isOpened():

        # Read feed
        ret, frame = cap.read()

        # Make detections
        image, results = mediapipe_detection(frame, holistic)
        print(results)
```

```
        # Draw landmarks
        draw_styled_landmarks(image, results)

        # Show to screen
        cv2.imshow('OpenCV Feed', image)

        # Break gracefully
        if cv2.waitKey(10) & 0xFF == ord('q'):
            break
    cap.release()
    cv2.destroyAllWindows()
len(results.left_hand_landmarks.landmark)
```

3. Extract Key point Values

```
pose = []
for res in results.pose_landmarks.landmark:
    test = np.array([res.x, res.y, res.z, res.visibility])
    pose.append(test) pose = np.array([[res.x, res.y, res.z, res.visibility] for res in results.pose_landmarks.landmark]).flatten() if results.pose_landmarks else np.zeros(132)
face = np.array([[res.x, res.y, res.z] for res in results.face_landmarks.landmark]).flatten() if results.face_landmarks else np.zeros(1404)
lh = np.array([[res.x, res.y, res.z] for res in results.left_hand_landmarks.landmark]).flatten() if results.left_hand_landmarks else np.zeros(21*3)
rh = np.array([[res.x, res.y, res.z] for res in results.right_hand_landmarks.landmark]).flatten() if results.right_hand_landmarks else np.zeros(21*3)
```

```
face = np.array([[res.x, res.y, res.z] for res in
results.face_landmarks.landmark]).flatten()

    if results.face_landmarks

    else np.zeros(1404)

def extract_keypoints(results):

    pose = np.array([[res.x, res.y, res.z, res.visibility] for res in
results.pose_landmarks.landmark]).flatten() if results.pose_landmarks else
np.zeros(33*4)

    face = np.array([[res.x, res.y, res.z] for res in
results.face_landmarks.landmark]).flatten() if results.face_landmarks else
np.zeros(468*3)

    lh = np.array([[res.x, res.y, res.z] for res in
results.left_hand_landmarks.landmark]).flatten() if results.left_hand_landmarks
else np.zeros(21*3)

    rh = np.array([[res.x, res.y, res.z] for res in
results.right_hand_landmarks.landmark]).flatten() if
results.right_hand_landmarks else np.zeros(21*3)

    return np.concatenate([pose, face, lh, rh])
```

4. Setup Folders for Collection

```
# Path for exported data, numpy arrays

DATA_PATH = os.path.join('MP_Data')


# Actions that we try to detect

actions = np.array(['hello', 'thanks', 'welcome'])


# Thirty videos worth of data

no_sequences = 30


# Videos are going to be 30 frames in length

sequence_length = 30
```

```
for action in actions:
    for sequence in range(no_sequences):
        try:
            os.makedirs(os.path.join(DATA_PATH, action, str(sequence)))
        except:
            pass
```

5. Collect Key point Values for Training and Testing

```
cap = cv2.VideoCapture(0)
# Set mediapipe model
with mp_holistic.Holistic(min_detection_confidence=0.5,
min_tracking_confidence=0.5) as holistic:


    # NEW LOOP
    # Loop through actions
    for action in actions:
        # Loop through sequences aka videos
        for sequence in range(no_sequences):
            # Loop through video length aka sequence length
            for frame_num in range(sequence_length):

                # Read feed
                ret, frame = cap.read()

                # Make detections
                image, results = mediapipe_detection(frame, holistic)
#                 print(results)

                # Draw landmarks
                draw_styled_landmarks(image, results)
```

```
            # NEW Apply wait logic
            if frame_num == 0:
                cv2.putText(image, 'STARTING COLLECTION', (120,200),
                            cv2.FONT_HERSHEY_SIMPLEX, 1, (0,255, 0), 4,
cv2.LINE_AA)
                cv2.putText(image, 'Collecting frames for {} Video Number
{}'.format(action, sequence), (15,12),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 1,
cv2.LINE_AA)
                # Show to screen
                cv2.imshow('OpenCV Feed', image)
                cv2.waitKey(2000)
            else:
                cv2.putText(image, 'Collecting frames for {} Video Number
{}'.format(action, sequence), (15,12),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 1,
cv2.LINE_AA)
                # Show to screen
                cv2.imshow('OpenCV Feed', image)

            # NEW Export keypoints
            keypoints = extract_keypoints(results)
            npy_path = os.path.join(DATA_PATH, action, str(sequence),
str(frame_num))
            np.save(npy_path, keypoints)

            # Break gracefully
            if cv2.waitKey(10) & 0xFF == ord('q'):
                break
```

cap.release()

cv2.destroyAllWindows()

```
cap.release()

cv2.destroyAllWindows()
```

6.Pre-process Data and Create Labels and Features

```
from sklearn.model_selection import train_test_split

from tensorflow.keras.utils import to_categorical

label_map = {label:num for num, label in enumerate(actions)}

sequences, labels = [], []

for action in actions:

    for sequence in range(no_sequences):

        window = []

        for frame_num in range(sequence_length):

            res = np.load(os.path.join(DATA_PATH, action, str(sequence), "{}.npy".format(frame_num)))

            window.append(res)

        sequences.append(window)

        labels.append(label_map[action])
```

7. Build and Train LSTM Neural Network

```
from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dense

from tensorflow.keras.callbacks import TensorBoard

log_dir = os.path.join('Logs')

tb_callback = TensorBoard(log_dir=log_dir)

model = Sequential()

model.add(LSTM(64, return_sequences=True, activation='relu', input_shape=(30,1662)))

model.add(LSTM(128, return_sequences=True, activation='relu'))

model.add(LSTM(64, return_sequences=False, activation='relu'))

model.add(Dense(64, activation='relu'))

model.add(Dense(32, activation='relu'))

model.add(Dense(actions.shape[0], activation='softmax'))
```

```
model.compile(optimizer='Adam', loss='categorical_crossentropy', metrics=['categorical_accuracy'])
model.fit(X_train, y_train, epochs=2000, callbacks=[tb_callback])
```

8.Save Models

```
model.save('action.h5')
model.load_weights('action.h5')
```

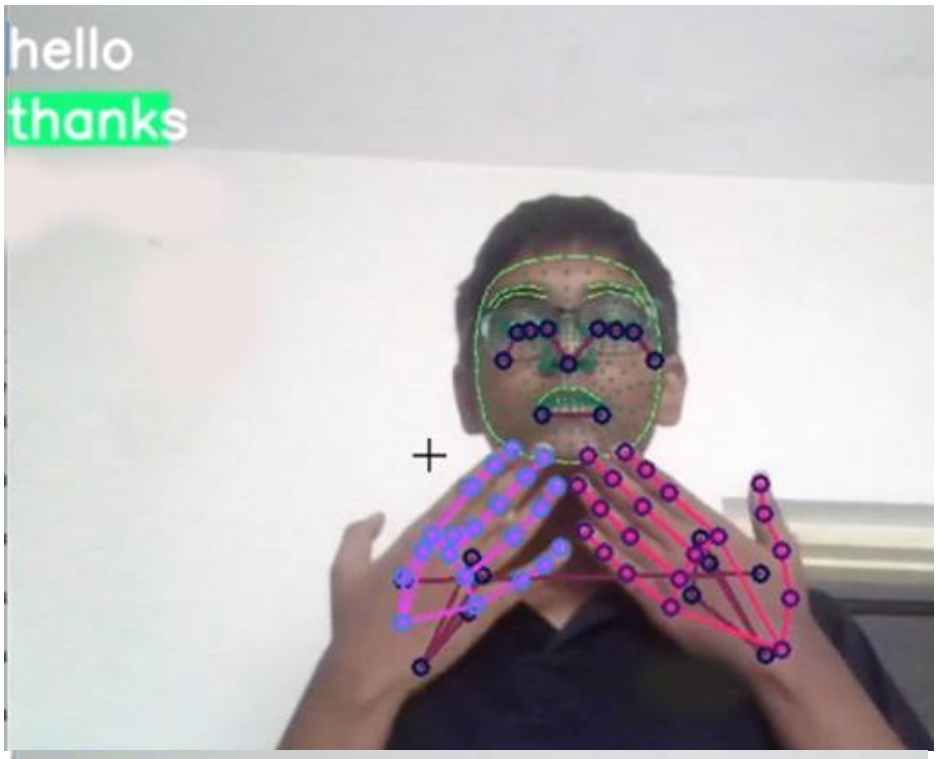
9.Test in Real Time

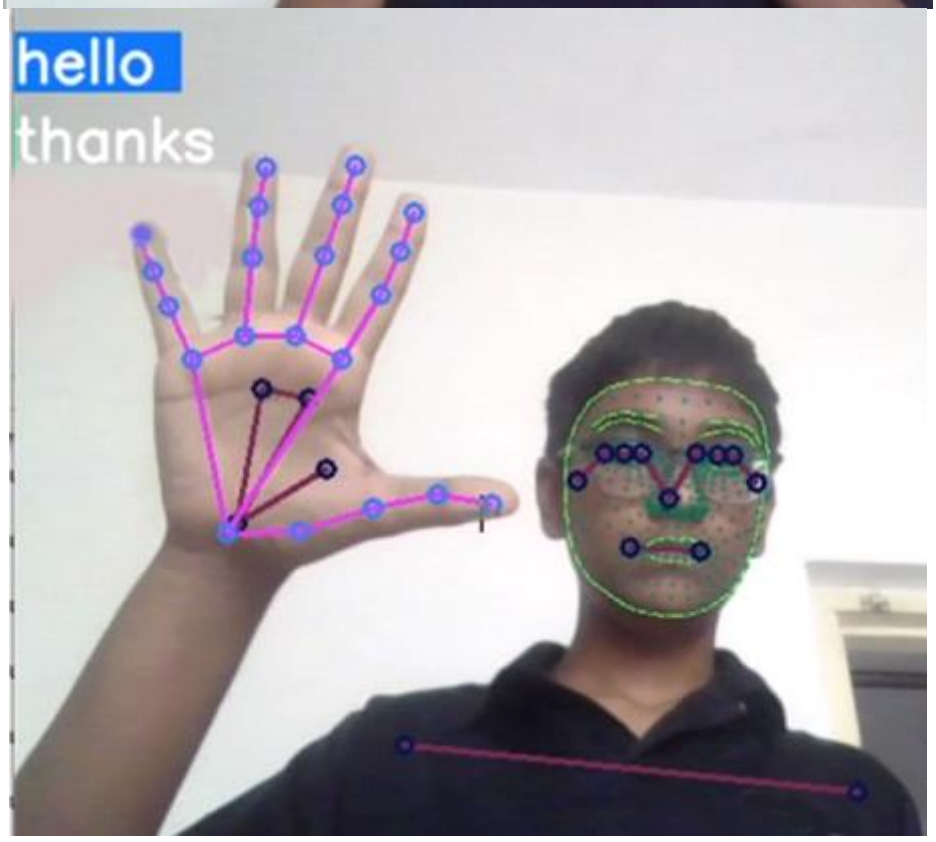
```
colors = [(245,117,16), (117,245,16), (16,117,245)]
def prob_viz(res, actions, input_frame, colors):
    output_frame = input_frame.copy()
    for num, prob in enumerate(res):
        cv2.rectangle(output_frame, (0,60+num*40), (int(prob*100), 90+num*40), colors[num], -1)
        cv2.putText(output_frame, actions[num], (0, 85+num*40), cv2.FONT_HERSHEY_SIMPLEX, 1, (255,255,255), 2, cv2.LINE_AA)

    return output_frame
sequence.reverse()
sequence.append('def')
# 1. New detection variables
sequence = []
sentence = []
threshold = 0.8

cap = cv2.VideoCapture(0)
# Set mediapipe model
with mp_holistic.Holistic(min_detection_confidence=0.5, min_tracking_confidence=0.5) as holistic:
    while cap.isOpened():

        # Read feed
        ret, frame = cap.read()

        # Make detections
```

```
        image, results = mediapipe_detection(frame, holistic)
        print(results)

        # Draw landmarks
        draw_styled_landmarks(image, results)

        # 2. Prediction logic
        keypoints = extract_keypoints(results)
#         sequence.insert(0,keypoints)
#         sequence = sequence[:30]
        sequence.append(keypoints)
        sequence = sequence[-30:]

        if len(sequence) == 30:
            res = model.predict(np.expand_dims(sequence, axis=0))[0]
            print(actions[np.argmax(res)])



        #3. Viz logic
            if res[np.argmax(res)] > threshold:
                if len(sentence) > 0:
                    if actions[np.argmax(res)] != sentence[-1]:
                        sentence.append(actions[np.argmax(res)])
                else:
                    sentence.append(actions[np.argmax(res)])

            if len(sentence) > 5:
                sentence = sentence[-5:]
            # Viz probabilities
            image = prob_viz(res, actions, image, colors)
```

```
        cv2.rectangle(image, (0,0), (640, 40), (245, 117, 16), -1)
        cv2.putText(image, ' '.join(sentence), (3,30),
                    cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2, cv2.LINE_AA)
            # Show to screen
        cv2.imshow('OpenCV Feed', image)

        # Break gracefully
        if cv2.waitKey(10) & 0xFF == ord('q'):
            break
    cap.release()
    cv2.destroyAllWindows()
cap.release()
cv2.destroyAllWindows()
```

6. Output:





7. Conclusion:

In this report, a functional real time vision based American sign language recognition for D&M people have been developed for asl alphabets. We achieved final accuracy of 90.5% on our dataset. We are able to detect almost all the symbols provided that they are shown properly, there is no noise in the background and lighting is adequate.

8.Reference:

Continuous Integration

- https://www.youtube.com/watch?v=yr23WyC2pr0.
- https://aclanthology.org/W16-6319.pdf
- https://www.cse.scu.edu/~mwang2/projects/NLP_English2IndianSignLanguage_18w.pdf
- https://youtu.be/S1Ow2D_DL0s

Unit testing

- https://cocalc.com/doc/jupyter-notebook.html
- https://opencv.org/