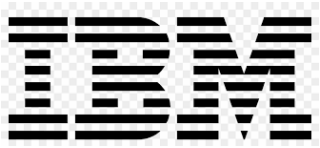
A Project Report On

# REAL TIME COMMUNICATION SYSTEM POWERED BY AI FOR SPACIALLY ABLED USING MACHINE LEARNING



**TEAM ID: PNT2022TMID50888**

## TEAM LEADER

RAHUL S

## TEAM MEMBERS

ASHICK RAHUMAN K

BALA VINOD S

SELVA SIVA S

# INDEX

# 1. INTRODUCTION

In this chapter we are seen about an overview of project aim, Objective and Background operation of environment of entire system.

Objectives are as follows:

- SIGN Language Translation.
- Alphabet Identification.
- Provide Corresponding output.

Operation Environment,

PROCESSOR - Intel Core Processor or AMD Processor.

OPERATING SYSTEM - Windows or Linux Distortions.

MEMORY - 4GB RAM or more

Libraries – OpenCV, Tensorflow, Keras, sklearn

This Project mainly created for Recognition sign Languages for specially abled people. It have a capturing window for recognition datum from specially abled persons hand gestures.

Problem Statement,

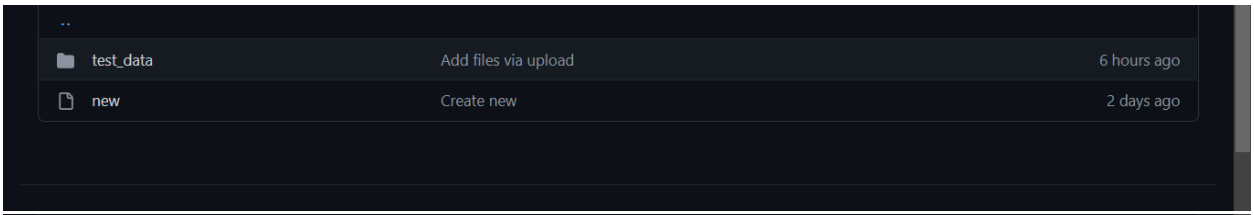
Specially abled people are not able to communicate with others and the surrounding environment. So we developed an AI system for these people to communicate with others. It more useful and effective technology for specially abled people.

Literature Survey,

| S. No | Paper Name | Journal Name | Description |
|---|---|---|---|
| 1. | Sign Language Recognition Application Systems for Deaf-Mute People | ScienceDirect | Every research has its own limitations and are still unable to be used commercially. |
| 2. | Sign language recognition: State of the art | ResearchGate.com | Sign language is used by deaf and hard hearing people to exchange information between their own community and with other people |
| 3. | Sign Language Recognition System using TensorFlow Object Detection API | arxiv.org | Communication is defined as the act of sharing or exchanging information, ideas or feelings. |

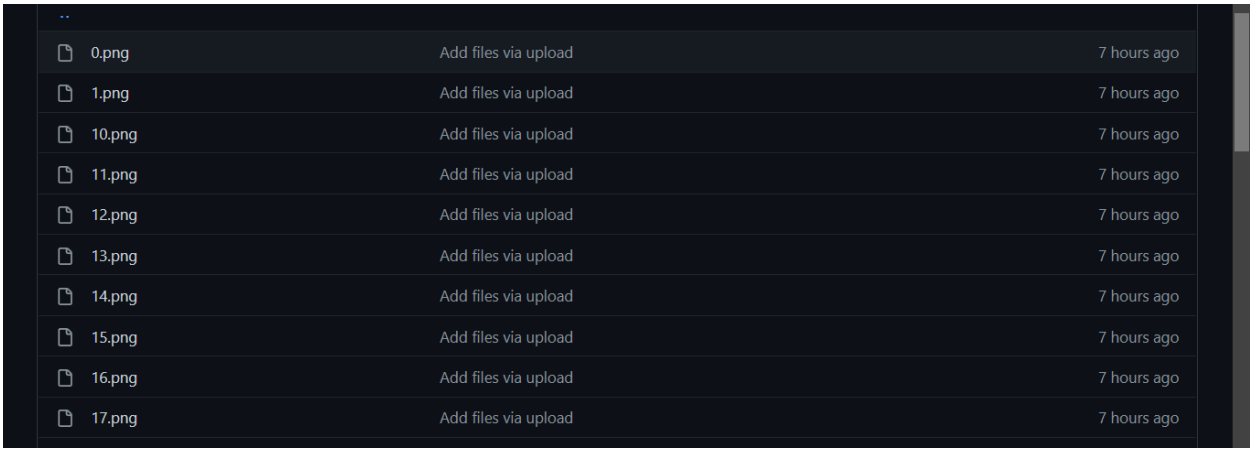
# 2. DATA COLLECTION

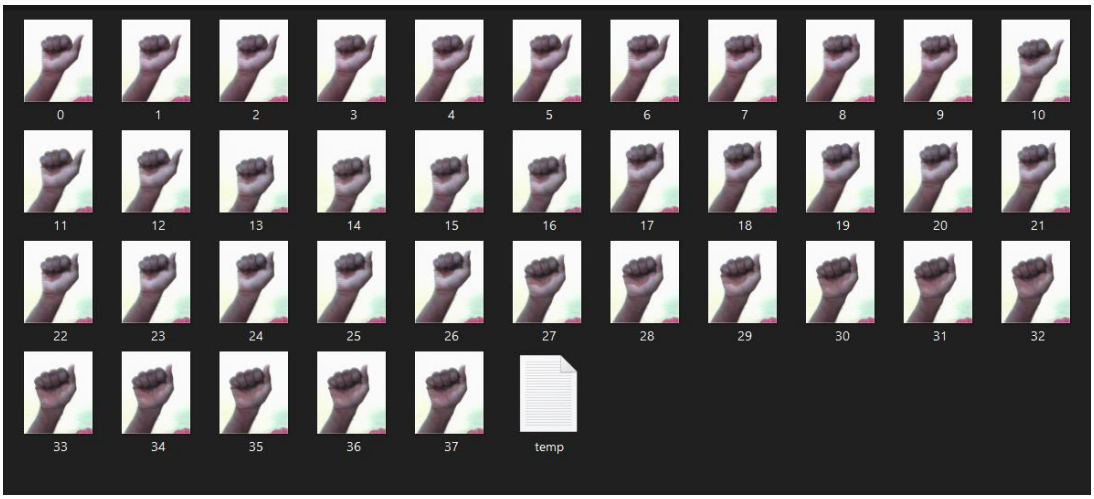
We are created a dataset based on hand gestures of persons. These datasets are separated by Alphabets order.





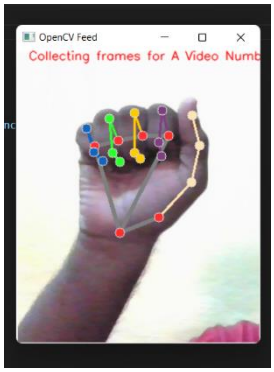
Data Arrangement Structure,



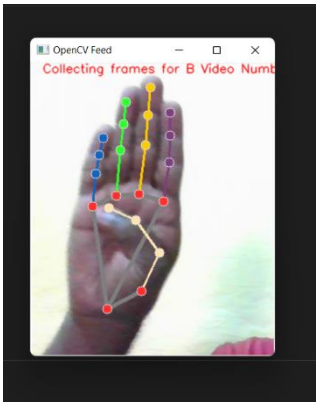



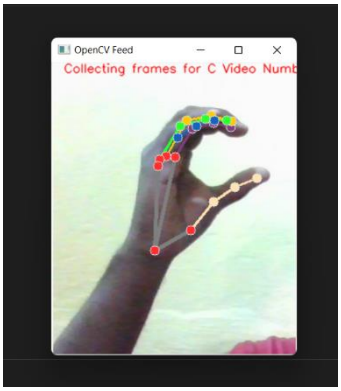
## 3. Image Pre-processing

Pre-processing is a common name for operations with images at the lowest level of abstraction — both input and output are intensity images. These iconic images are of the same kind as the original data captured by the sensor, with an intensity image usually represented by a matrix of image function values.




Alphabet A



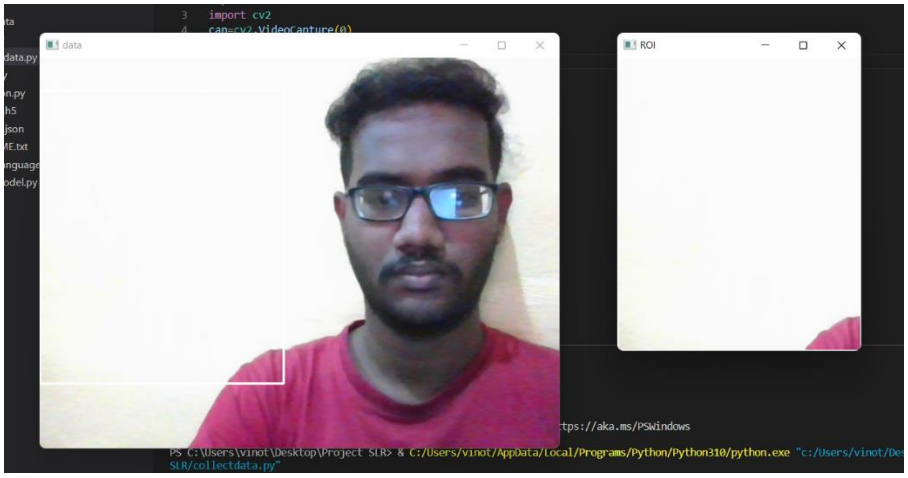

Alphabet B

Alphabet C

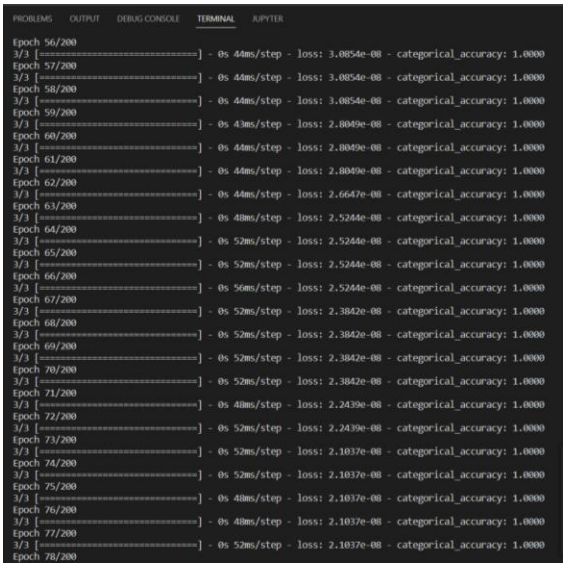
These same data pre-processing step followed for all other alphabets in Image Pre-processing phase.

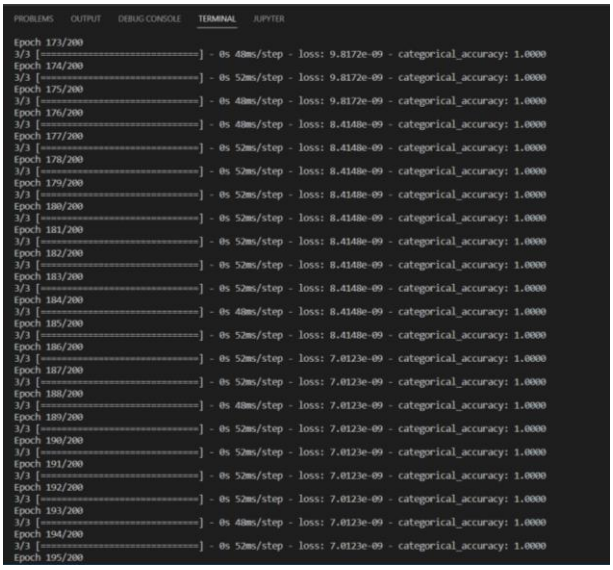
# 4. MODEL BUILDING

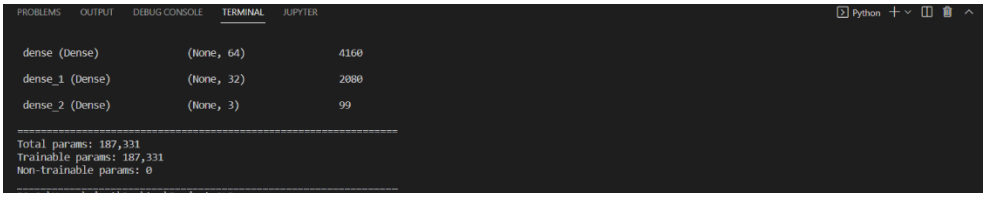
The process that covers right from source data identification to model development, model deployment and model maintenance.




Developing a Model based on collected data,



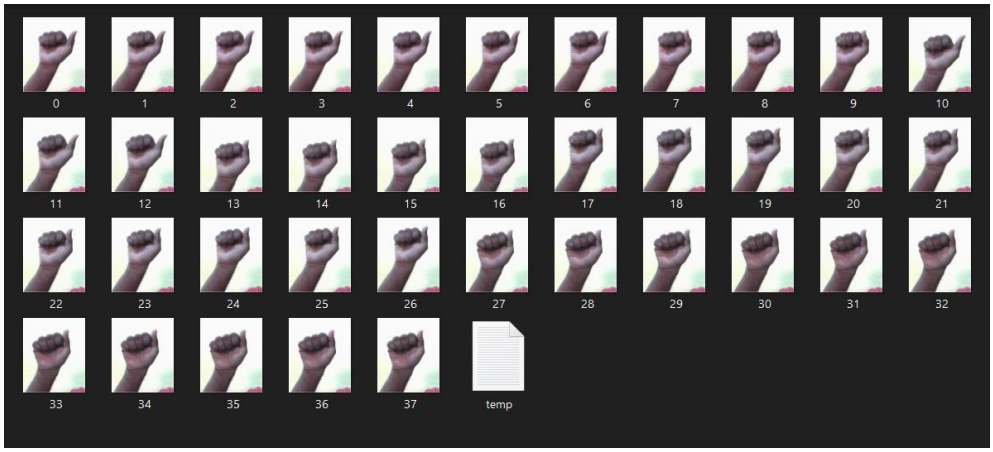



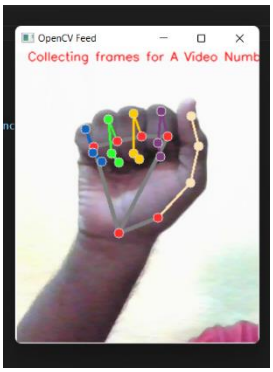



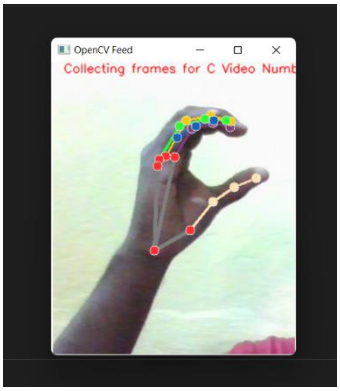
# 5. TRAIN and TEST

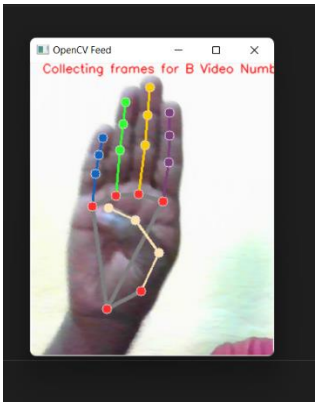
The Process which takes place for training a generated model with corresponding epoch and collected data, In this case tested are done by few collected data.

trained data,



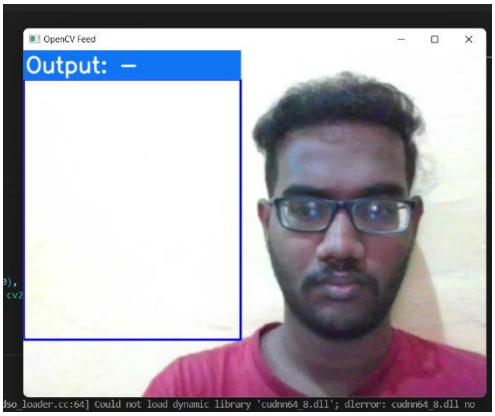



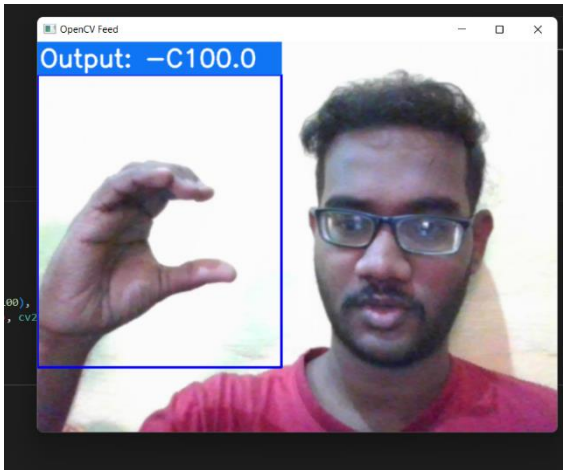

Black box Testing,

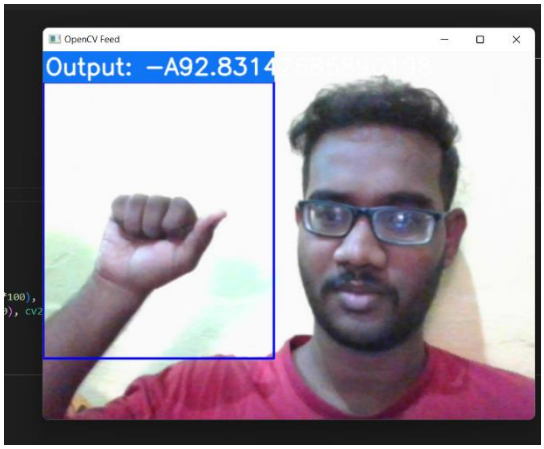
Is a software testing method in which the functionalities of software applications are tested without having knowledge of internal code structure, implementation details and internal paths. Black Box Testing mainly focuses on input and output of software applications and it is entirely based on software requirements and specifications.

# 6. OUTPUT




Initial Live Window




## ### SOURCE CODE

**App.py**

```
from function import *
from keras.utils import to_categorical
from keras.models import model_from_json
from keras.layers import LSTM, Dense
from keras.callbacks import TensorBoard
json_file = open("model.json", "r")
model_json = json_file.read()
json_file.close()
model = model_from_json(model_json)
model.load_weights("model.h5")

colors = []
for i in range(0,20):
    colors.append((245,117,16))
print(len(colors))
def prob_viz(res, actions, input_frame, colors,threshold):
    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
```

```
# 1. New detection variables
sequence = []
sentence = []
accuracy=[]
predictions = []
threshold = 0.8

cap = cv2.VideoCapture(0)
# cap = cv2.VideoCapture("https://192.168.43.41:8080/video")
# Set mediapipe model
with mp_hands.Hands(
    model_complexity=0,
    min_detection_confidence=0.5,
    min_tracking_confidence=0.5) as hands:
    while cap.isOpened():

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

        # Make detections
        cropframe=frame[40:400,0:300]
        # print(frame.shape)
        frame=cv2.rectangle(frame,(0,40),(300,400),255,2)
        # frame=cv2.putText(frame,"Active
Region",(75,25),cv2.FONT_HERSHEY_COMPLEX_SMALL,2,255,2)
        image, results = mediapipe_detection(cropframe, hands)
        # print(results)
```

```
# Draw landmarks
# draw_styled_landmarks(image, results)
# 2. Prediction logic
keypoints = extract_keypoints(results)
sequence.append(keypoints)
sequence = sequence[-30:]

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



    #3. Viz logic
        if np.unique(predictions[-10:])[0]==np.argmax(res):
            if res[np.argmax(res)] > threshold:
                if len(sentence) > 0:
                    if actions[np.argmax(res)] != sentence[-1]:
                        sentence.append(actions[np.argmax(res)])
                        accuracy.append(str(res[np.argmax(res)]*100))
                else:
                    sentence.append(actions[np.argmax(res)])
                    accuracy.append(str(res[np.argmax(res)]*100))

        if len(sentence) > 1:
            sentence = sentence[-1:]
            accuracy=accuracy[-1:]
```

```
            # Viz probabilities
            # frame = prob_viz(res, actions, frame, colors,threshold)
        except Exception as e:
            # print(e)
            pass

        cv2.rectangle(frame, (0,0), (300, 40), (245, 117, 16), -1)
        cv2.putText(frame,"Output: -"+' '.join(sentence)+".join(accuracy), (3,30),
                    cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2, cv2.LINE_AA)

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

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

**collectdata.py**

```
import os
import cv2
cap=cv2.VideoCapture(0)
```

```
directory='Image/'
while True:
    _,frame=cap.read()
    count = {
            'a': len(os.listdir(directory+"/A")),
            'b': len(os.listdir(directory+"/B")),
            'c': len(os.listdir(directory+"/C")),
            'd': len(os.listdir(directory+"/D")),
            'e': len(os.listdir(directory+"/E")),
            'f': len(os.listdir(directory+"/F")),
            'g': len(os.listdir(directory+"/G")),
            'h': len(os.listdir(directory+"/H")),
            'i': len(os.listdir(directory+"/I")),
            'j': len(os.listdir(directory+"/J")),
            'k': len(os.listdir(directory+"/K")),
            'l': len(os.listdir(directory+"/L")),
            'm': len(os.listdir(directory+"/M")),
            'n': len(os.listdir(directory+"/N")),
            'o': len(os.listdir(directory+"/O")),
            'p': len(os.listdir(directory+"/P")),
            'q': len(os.listdir(directory+"/Q")),
            'r': len(os.listdir(directory+"/R")),
            's': len(os.listdir(directory+"/S")),
            't': len(os.listdir(directory+"/T")),
            'u': len(os.listdir(directory+"/U")),
            'v': len(os.listdir(directory+"/V")),
            'w': len(os.listdir(directory+"/W")),
            'x': len(os.listdir(directory+"/X")),
```

```
        'y': len(os.listdir(directory+"/Y")),
        'z': len(os.listdir(directory+"/Z"))
        }
    # cv2.putText(frame, "a : "+str(count['a']), (10, 100), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
    # cv2.putText(frame, "b : "+str(count['b']), (10, 110), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
    # cv2.putText(frame, "c : "+str(count['c']), (10, 120), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
    # cv2.putText(frame, "d : "+str(count['d']), (10, 130), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
    # cv2.putText(frame, "e : "+str(count['e']), (10, 140), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
    # cv2.putText(frame, "f : "+str(count['f']), (10, 150), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
    # cv2.putText(frame, "g : "+str(count['g']), (10, 160), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
    # cv2.putText(frame, "h : "+str(count['h']), (10, 170), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
    # cv2.putText(frame, "i : "+str(count['i']), (10, 180), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
    # cv2.putText(frame, "k : "+str(count['k']), (10, 190), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
    # cv2.putText(frame, "l : "+str(count['l']), (10, 200), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
    # cv2.putText(frame, "m : "+str(count['m']), (10, 210), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
    # cv2.putText(frame, "n : "+str(count['n']), (10, 220), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
    # cv2.putText(frame, "o : "+str(count['o']), (10, 230), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
    # cv2.putText(frame, "p : "+str(count['p']), (10, 240), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
```

```
    # cv2.putText(frame, "q : "+str(count['q']), (10, 250), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)

    # cv2.putText(frame, "r : "+str(count['r']), (10, 260), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)

    # cv2.putText(frame, "s : "+str(count['s']), (10, 270), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)

    # cv2.putText(frame, "t : "+str(count['t']), (10, 280), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)

    # cv2.putText(frame, "u : "+str(count['u']), (10, 290), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)

    # cv2.putText(frame, "v : "+str(count['v']), (10, 300), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)

    # cv2.putText(frame, "w : "+str(count['w']), (10, 310), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)

    # cv2.putText(frame, "x : "+str(count['x']), (10, 320), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)

    # cv2.putText(frame, "y : "+str(count['y']), (10, 330), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)

    # cv2.putText(frame, "z : "+str(count['z']), (10, 340), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)

    row = frame.shape[1]

    col = frame.shape[0]

    cv2.rectangle(frame,(0,40),(300,400),(255,255,255),2)

    cv2.imshow("data",frame)

    cv2.imshow("ROI",frame[40:400,0:300])

    frame=frame[40:400,0:300]

    interrupt = cv2.waitKey(10)

    if interrupt & 0xFF == ord('a'):

        cv2.imwrite(directory+'A/'+str(count['a'])+'.png',frame)

    if interrupt & 0xFF == ord('b'):

        cv2.imwrite(directory+'B/'+str(count['b'])+'.png',frame)

    if interrupt & 0xFF == ord('c'):
```

```
    cv2.imwrite(directory+'C/'+str(count['c'])+'.png',frame)
if interrupt & 0xFF == ord('d'):
    cv2.imwrite(directory+'D/'+str(count['d'])+'.png',frame)
if interrupt & 0xFF == ord('e'):
    cv2.imwrite(directory+'E/'+str(count['e'])+'.png',frame)
if interrupt & 0xFF == ord('f'):
    cv2.imwrite(directory+'F/'+str(count['f'])+'.png',frame)
if interrupt & 0xFF == ord('g'):
    cv2.imwrite(directory+'G/'+str(count['g'])+'.png',frame)
if interrupt & 0xFF == ord('h'):
    cv2.imwrite(directory+'H/'+str(count['h'])+'.png',frame)
if interrupt & 0xFF == ord('i'):
    cv2.imwrite(directory+'I/'+str(count['i'])+'.png',frame)
if interrupt & 0xFF == ord('j'):
    cv2.imwrite(directory+'J/'+str(count['j'])+'.png',frame)
if interrupt & 0xFF == ord('k'):
    cv2.imwrite(directory+'K/'+str(count['k'])+'.png',frame)
if interrupt & 0xFF == ord('l'):
    cv2.imwrite(directory+'L/'+str(count['l'])+'.png',frame)
if interrupt & 0xFF == ord('m'):
    cv2.imwrite(directory+'M/'+str(count['m'])+'.png',frame)
if interrupt & 0xFF == ord('n'):
    cv2.imwrite(directory+'N/'+str(count['n'])+'.png',frame)
if interrupt & 0xFF == ord('o'):
    cv2.imwrite(directory+'O/'+str(count['o'])+'.png',frame)
if interrupt & 0xFF == ord('p'):
    cv2.imwrite(directory+'P/'+str(count['p'])+'.png',frame)
if interrupt & 0xFF == ord('q'):
```

```
        cv2.imwrite(directory+'Q/'+str(count['q'])+'.png',frame)
    if interrupt & 0xFF == ord('r'):
        cv2.imwrite(directory+'R/'+str(count['r'])+'.png',frame)
    if interrupt & 0xFF == ord('s'):
        cv2.imwrite(directory+'S/'+str(count['s'])+'.png',frame)
    if interrupt & 0xFF == ord('t'):
        cv2.imwrite(directory+'T/'+str(count['t'])+'.png',frame)
    if interrupt & 0xFF == ord('u'):
        cv2.imwrite(directory+'U/'+str(count['u'])+'.png',frame)
    if interrupt & 0xFF == ord('v'):
        cv2.imwrite(directory+'V/'+str(count['v'])+'.png',frame)
    if interrupt & 0xFF == ord('w'):
        cv2.imwrite(directory+'W/'+str(count['w'])+'.png',frame)
    if interrupt & 0xFF == ord('x'):
        cv2.imwrite(directory+'X/'+str(count['x'])+'.png',frame)
    if interrupt & 0xFF == ord('y'):
        cv2.imwrite(directory+'Y/'+str(count['y'])+'.png',frame)
    if interrupt & 0xFF == ord('z'):
        cv2.imwrite(directory+'Z/'+str(count['z'])+'.png',frame)


cap.release()
cv2.destroyAllWindows()
```

**data.py**

```
from function import *
from time import sleep

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

# cap = cv2.VideoCapture(0)
# Set mediapipe model
with mp_hands.Hands(
    model_complexity=0,
    min_detection_confidence=0.5,
    min_tracking_confidence=0.5) as hands:

    # 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()
                frame=cv2.imread('Image/{}/{}.png'.format(action,sequence))
```

```
            # frame=cv2.imread('{}{}.png'.format(action,sequence))
            # frame=cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)

            # Make detections
            image, results = mediapipe_detection(frame, hands)
#               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(200)
            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()
```

**function.py**

```
import cv2
import numpy as np
import os
import mediapipe as mp

mp_drawing = mp.solutions.drawing_utils
mp_drawing_styles = mp.solutions.drawing_styles
mp_hands = mp.solutions.hands

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_styled_landmarks(image, results):
    if results.multi_hand_landmarks:
      for hand_landmarks in results.multi_hand_landmarks:
        mp_drawing.draw_landmarks(
            image,
            hand_landmarks,
            mp_hands.HAND_CONNECTIONS,
            mp_drawing_styles.get_default_hand_landmarks_style(),
            mp_drawing_styles.get_default_hand_connections_style())



def extract_keypoints(results):
    if results.multi_hand_landmarks:
      for hand_landmarks in results.multi_hand_landmarks:
        rh = np.array([[res.x, res.y, res.z] for res in hand_landmarks.landmark]).flatten() if hand_landmarks else np.zeros(21*3)
        return(np.concatenate([rh]))
# Path for exported data, numpy arrays
DATA_PATH = os.path.join('MP_Data')


actions = np.array(['A','B','C'])


no_sequences = 30


sequence_length = 30
```

**trainmodel.py**

```
from function import *
from sklearn.model_selection import train_test_split
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import LSTM, Dense
from keras.callbacks import TensorBoard
label_map = {label:num for num, label in enumerate(actions)}
# print(label_map)
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])

X = np.array(sequences)
y = to_categorical(labels).astype(int)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.05)

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=(30,63)))
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'))
res = [.7, 0.2, 0.1]

model.compile(optimizer='Adam', loss='categorical_crossentropy',
metrics=['categorical_accuracy'])
model.fit(X_train, y_train, epochs=200, callbacks=[tb_callback])
model.summary()

model_json = model.to_json()
with open("model.json", "w") as json_file:
    json_file.write(model_json)
model.save('model.h5')
```