```python
#Data Collecting Python File -> collectdata.py


#----------------------------> Importing important libraries <----------------------------
import os
import cv2


#-----------------------------> Image collection using 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"))
             }

    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'):
```

```
61          cv2.imwrite(directory+'F/'+str(count['f'])+'.png',frame)
62      if interrupt & 0xFF == ord('g'):
63          cv2.imwrite(directory+'G/'+str(count['g'])+'.png',frame)
64      if interrupt & 0xFF == ord('h'):
65          cv2.imwrite(directory+'H/'+str(count['h'])+'.png',frame)
66      if interrupt & 0xFF == ord('i'):
67          cv2.imwrite(directory+'I/'+str(count['i'])+'.png',frame)
68      if interrupt & 0xFF == ord('j'):
69          cv2.imwrite(directory+'J/'+str(count['j'])+'.png',frame)
70      if interrupt & 0xFF == ord('k'):
71          cv2.imwrite(directory+'K/'+str(count['k'])+'.png',frame)
72      if interrupt & 0xFF == ord('l'):
73          cv2.imwrite(directory+'L/'+str(count['l'])+'.png',frame)
74      if interrupt & 0xFF == ord('m'):
75          cv2.imwrite(directory+'M/'+str(count['m'])+'.png',frame)
76      if interrupt & 0xFF == ord('n'):
77          cv2.imwrite(directory+'N/'+str(count['n'])+'.png',frame)
78      if interrupt & 0xFF == ord('o'):
79          cv2.imwrite(directory+'O/'+str(count['o'])+'.png',frame)
80      if interrupt & 0xFF == ord('p'):
81          cv2.imwrite(directory+'P/'+str(count['p'])+'.png',frame)
82      if interrupt & 0xFF == ord('q'):
83          cv2.imwrite(directory+'Q/'+str(count['q'])+'.png',frame)
84      if interrupt & 0xFF == ord('r'):
85          cv2.imwrite(directory+'R/'+str(count['r'])+'.png',frame)
86      if interrupt & 0xFF == ord('s'):
87          cv2.imwrite(directory+'S/'+str(count['s'])+'.png',frame)
88      if interrupt & 0xFF == ord('t'):
89          cv2.imwrite(directory+'T/'+str(count['t'])+'.png',frame)
90      if interrupt & 0xFF == ord('u'):
91          cv2.imwrite(directory+'U/'+str(count['u'])+'.png',frame)
92      if interrupt & 0xFF == ord('v'):
93          cv2.imwrite(directory+'V/'+str(count['v'])+'.png',frame)
94      if interrupt & 0xFF == ord('w'):
95          cv2.imwrite(directory+'W/'+str(count['w'])+'.png',frame)
96      if interrupt & 0xFF == ord('x'):
97          cv2.imwrite(directory+'X/'+str(count['x'])+'.png',frame)
98      if interrupt & 0xFF == ord('y'):
99          cv2.imwrite(directory+'Y/'+str(count['y'])+'.png',frame)
100     if interrupt & 0xFF == ord('z'):
101         cv2.imwrite(directory+'Z/'+str(count['z'])+'.png',frame)
102
103
104 cap.release()
105 cv2.destroyAllWindows()
```

```python
#File containing common functions used -> function.py

#---------------------------------> Importing important libraries <---------------------
import cv2
import numpy as np
import os
import mediapipe as mp


#---------------------------------> Some other utilities <-----------------------------
mp_drawing = mp.solutions.drawing_utils
mp_drawing_styles = mp.solutions.drawing_styles
mp_hands = mp.solutions.hands


#---------------------------------> Defining all functions <---------------------------

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]))


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

actions = np.array(['A','B','C','D','E','F','G','H','I'])

no_sequences = 30

sequence_length = 30


```

```python
#File converting landmarks of hands into .np files  -> data.py


#--------------------------> Importing Important libraries <------------------------
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


# Setting the mediapipe model
with mp_hands.Hands(
    model_complexity=0,
    min_detection_confidence=0.5,
    min_tracking_confidence=0.5) as hands:

    # Looping  through actions
    for action in actions:
        # Looping through sequences i.e videos
        for sequence in range(no_sequences):
            # Loop through video length i.e sequence length
            for frame_num in range(sequence_length):

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

                # Making detections
                image, results = mediapipe_detection(frame, hands)
#
                # Drawing landmarks
                draw_styled_landmarks(image, results)

                # Now Applying 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)

                    # To show on 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)
                    # To show on screen
                    cv2.imshow('OpenCV Feed', image)

                # Some 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)

                # Breaking  gracefully
```

```
60                 if cv2.waitKey(10) & 0xFF == ord('q'):
61                     break
62 
63     cv2.destroyAllWindows()
64 
65 
66 
```

```python
#Training model file -> trainmodel.py

#---------------------------------> Importing important libraries <----------------------

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



#---------------------------------------> Training Logic <------------------------------

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])

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')
```

```python
#Main application file -> app.py

#--------------------------> Importing important libraries <-----------------------------
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




#-------------------------> Defining important files and locations <---------------------

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


# ------------------------------> New detection variables <-----------------------------
sequence = []
sentence = []
accuracy=[]
predictions = []
threshold = 0.8

cap = cv2.VideoCapture(0)




#--------------------------------> Setting mediapipe model <-------------------------
with mp_hands.Hands(
    model_complexity=0,
    min_detection_confidence=0.5,
    min_tracking_confidence=0.5) as hands:
    while cap.isOpened():



#-----------------------------> Detection logic <-------------------------------------
        # Reading of the feed
```

```
        ret, frame = cap.read()

        # Making detections
        cropframe=frame[40:400,0:300]


        frame=cv2.rectangle(frame,(0,40),(300,400),255,2)
        image, results = mediapipe_detection(cropframe, hands)



#------------------------------> 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))


            #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:]


        except Exception as 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)

        cv2.imshow('OpenCV Feed', frame)

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