**1. Import and Install Dependencies**

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

[1]

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

**2. Keypoints using MP Holistic**

[2]

mp\_holistic = mp.solutions.holistic *# Holistic model*  
mp\_drawing = mp.solutions.drawing\_utils *# Drawing utilities*

[3]

**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

[4]

**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*

[5]

**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)  
                             )

[6]

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) & **0x**FF == ord(**'q'**):  
            **break**  
    cap.release()  
    cv2.destroyAllWindows()

<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>  
<class 'mediapipe.python.solution\_base.SolutionOutputs'>

[7]

draw\_styled\_landmarks(frame, results)

[8]

plt.imshow(cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB))

<matplotlib.image.AxesImage at 0x1ca28fda860>



**3. Extract Keypoint Values**

[9]

**if** (results.left\_hand\_landmarks != **None**):  
    print(len(results.left\_hand\_landmarks.landmark))  
**elif**(results.right\_hand\_landmarks != **None**):  
    print(len(results.right\_hand\_landmarks.landmark))

21

[10]

pose = []  
**for** res **in** results.pose\_landmarks.landmark:  
    test = np.array([res.x, res.y, res.z, res.visibility])  
    pose.append(test)

[11]

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)

[12]

**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])

[13]

result\_test = extract\_keypoints(results)

[14]

result\_test

array([ 0.33045992, 0.50875646, -0.77007359, ..., 0. ,  
 0. , 0. ])

[15]

np.save(**'0'**, result\_test)

[22]

np.load(**'0.npy'**)

array([ 0.6034317 , 0.44217509, -1.00870204, ..., 0.4556205 ,  
 0.27668288, -0.03398165])

**4. Setup Folders for Collection**

[13]

*# Path for exported data, numpy arrays*  
DATA\_PATH = os.path.join(**'MP\_Data'**)   
  
*# Actions that we try to detect*  
actions = np.array([**'again'**, **'ASL'**, **'yes'**])  
  
*# Thirty videos worth of data*  
no\_sequences =  40  
  
*# Videos are going to be 30 frames in length*  
sequence\_length = 30  
  
*# Folder start*  
start\_folder = 0

[52]

**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 Keypoint Values for Training and Testing**

[14]

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(start\_folder, start\_folder+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)  
  
                *# 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, **'Frames for {} Video Number {}'**.format(action, sequence), (15,12),   
                               cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, (255, 0, 0), 1, cv2.LINE\_AA)  
                    *# Show to screen*  
                    cv2.imshow(**'OpenCV Feed'**, image)  
                    cv2.waitKey(500)  
                **else**:   
                    cv2.putText(image, **'Frames for {} Video Number {}'**.format(action, sequence), (15,12),   
                               cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, (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) & **0x**FF == ord(**'q'**):  
                    **break**  
                      
    cap.release()  
    cv2.destroyAllWindows()

[1]

cap.release()  
cv2.destroyAllWindows()

**---------------------------------------------------------------------------**  
**NameError** Traceback (most recent call last)  
**<ipython-input-1-37d044a27ce5>** in <module> **----> 1**cap**.**release**()** 2 cv2**.**destroyAllWindows**()**  
**NameError**: name 'cap' is not defined

**6. Preprocess Data and Create Labels and Features**

[16]

**from** sklearn.model\_selection **import** train\_test\_split  
**from** tensorflow.keras.utils **import** to\_categorical

[21]

*# Path for exported data, numpy arrays*  
DATA\_PATH = os.path.join(**'MP\_Data'**)   
  
*# Actions that we try to detect*  
actions = np.array([**'hello'**, **'thanks'**, **'iloveyou'**,**'again'**, **'ASL'**, **'yes'**])  
  
*# Thirty videos worth of data*  
no\_sequences =  40  
  
*# Videos are going to be 30 frames in length*  
sequence\_length = 30  
  
*# Folder start*  
start\_folder = 0

[18]

label\_map = {label:num **for** num, label **in** enumerate(actions)}

[19]

label\_map

{'hello': 0, 'thanks': 1, 'iloveyou': 2, 'again': 3, 'ASL': 4, 'yes': 5}

[22]

sequences, labels = [], []  
**for** action **in** actions:  
    **for** sequence **in** np.array(os.listdir(os.path.join(DATA\_PATH, action))).astype(int):  
        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])

[23]

np.array(sequences).shape

(240, 30, 1662)

[24]

np.array(labels).shape

(240,)

[25]

X = np.array(sequences)

[26]

X.shape

(240, 30, 1662)

[27]

y = to\_categorical(labels).astype(int)

[28]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.1)

[29]

y\_test.shape

(24, 6)

[33]

actions.shape[0]

6