[1]:
```
import sys
print(sys.version)
```

```
3.11.7 | packaged by Anaconda, Inc. | (main, Dec 15 2023, 18:05:47) [MSC v.1916
64 bit (AMD64)]
```

1 Import and Intallation of Dependencies

[ ]:
```
!pip install tensorflow opencv-python mediapipe scikit-learn matplotlib
```

[1]:
```
import cv2
import numpy as np
import os
from matplotlib import pyplot as plt
import time
import mediapipe as mp
```

2 Marking Keypoints using Mediapipe Holistics

[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):
    # Check and draw face landmarks
    if results.face_landmarks:
        mp_drawing.draw_landmarks(image, results.face_landmarks, mp_holistic.
 ↪FACEMESH_TESSELATION)

    # Check and draw pose landmarks
    if results.pose_landmarks:
        mp_drawing.draw_landmarks(image, results.pose_landmarks, mp_holistic.
 ↪POSE_CONNECTIONS)

    # Check and draw left hand landmarks
    if results.left_hand_landmarks:
        mp_drawing.draw_landmarks(image, results.left_hand_landmarks,␣
 ↪mp_holistic.HAND_CONNECTIONS)
```

```
    # Check and draw right hand landmarks
    if results.right_hand_landmarks:
        mp_drawing.draw_landmarks(image, results.right_hand_landmarks,␣
↪mp_holistic.HAND_CONNECTIONS)
```

[5]:
```
def draw_styled_landmarks(image, results):
    # Draw styled face landmarks if present
    if results.face_landmarks:
        mp_drawing.draw_landmarks(
            image, results.face_landmarks, mp_holistic.FACEMESH_CONTOURS,
            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 styled pose landmarks if present
    if results.pose_landmarks:
        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 styled left hand landmarks if present
    if results.left_hand_landmarks:
        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 styled right hand landmarks if present
    if results.right_hand_landmarks:
        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)
        )
```

```
# Note: Ensure the 'results' object is correctly populated with landmarks for␣
↪face, pose, and hands before calling this function.
```

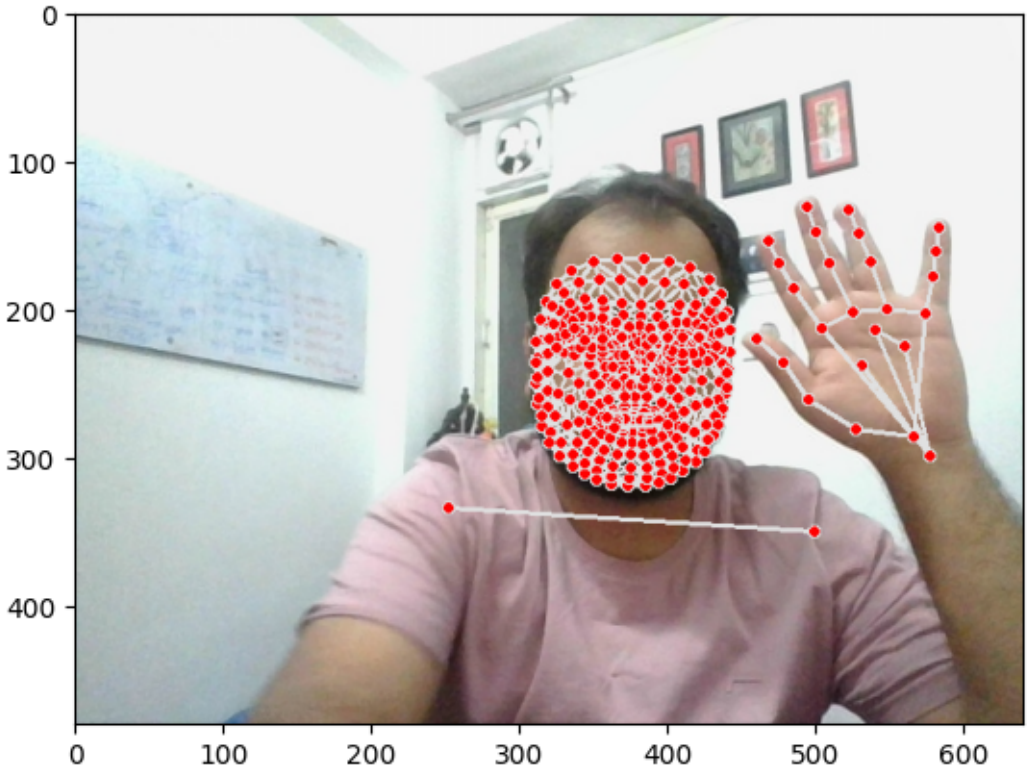
[6]:
```
cap = cv2.VideoCapture(0)
# Set mediapipe model
with mp_holistic.Holistic(min_detection_confidence=0.5,␣
↪min_tracking_confidence=0.5) as holistic:
    while cap.isOpened():

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

```
<class 'mediapipe.python.solution_base.SolutionOutputs'>
```

[9]:
```
draw_landmarks(frame, results)
```

[10]:
```
plt.imshow(cv2.cvtColor(frame, cv2.COLOR_BGR2RGB))
```

[10]:
```
<matplotlib.image.AxesImage at 0x1f9f6bb9150>
```



3 Extracting Key point Vaues

[11]:
```
pose = []
for res in results.pose_landmarks.landmark:
    test = np.array([res.x, res.y, res.z, res.visibility])
    pose.append(test)
```

[12]:
```
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)
```

[13]:
```
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])
```

[14]:
```
result_test = extract_keypoints(results)
```

[15]:
```
result_test
```

[15]:
```
array([ 0.60258186,  0.51626396, -0.67121172, ...,  0.        ,
        0.        ,  0.        ])
```

4 Setting Up folders for data collection

[16]:
```
# Path for exported data, numpy arrays
DATA_PATH = os.path.join('MP_Data')

# Actions that we try to detect
actions = np.
↪array(['Deaf','Father','Friend','Goodluck','Hello','Love','Mother','No','Peace','Please','Tha

# Thirty videos worth of data
no_sequences = 30

# Videos are going to be 30 frames in length
sequence_length = 30
```

[17]:
```
# hello
 ## 0
 ## 1
 ## 2
 ## ...
 ## 29
# thanks

# I love you
```

[18]:
```
for action in actions:
    for sequence in range(no_sequences):
        try:
            os.makedirs(os.path.join(DATA_PATH, action, str(sequence)))
        except:
            pass
```

5 Collecting Keypoint values for training and testing

[165]:
```
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()
```

[19]:
```
cap.release()
cv2.destroyAllWindows()
```

6 Pre-processing Data

[20]:
```
import numpy as np
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.utils import to_categorical
```

[21]:
```
label_map = {label:num for num, label in enumerate(actions)}
```

[22]:
```
label_map
```

[22]:
```
{'Deaf': 0,
 'Father': 1,
 'Friend': 2,
 'Goodluck': 3,
 'Hello': 4,
 'Love': 5,
 'Mother': 6,
 'No': 7,
 'Peace': 8,
 'Please': 9,
 'Thanks': 10,
 'Washroom': 11,
 'Yes': 12}
```

[23]:
```
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])
```

[24]:
```
np.array(sequences).shape
```

[24]:
```
(390, 30, 1662)
```

[25]:
```
np.array(labels).shape
```

[25]:
```
(390,)
```

[26]:
```
X = np.array(sequences)
```

[27]:
```
X.shape
```

[27]:
```
(390, 30, 1662)
```

[28]:
```
y = to_categorical(labels).astype(int)
```

[29]:
```
y
```

[29]:
```
array([[1, 0, 0, ..., 0, 0, 0],
       [1, 0, 0, ..., 0, 0, 0],
       [1, 0, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 1],
       [0, 0, 0, ..., 0, 0, 1],
       [0, 0, 0, ..., 0, 0, 1]])
```

[30]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.05)
```

[31]:
```
y_train.shape
```

[31]:
```
(370, 13)
```

7 Training of LSTM (Long short-term memory) Neural Network Model

[32]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from tensorflow.keras.callbacks import TensorBoard
```

[33]:
```
log_dir = os.path.join('Logs')
tb_callback = TensorBoard(log_dir=log_dir)
```

[34]:
```
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'))
```

```
I:\Users\bhavishya\desktop\_ _ _\venvone\Lib\site-
packages\keras\src\layers\rnn\rnn.py:204: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
```

[35]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Input
model = Sequential()
model.add(Input(shape=(30, 1662)))  # Define the input shape explicitly with an␣
 ↪Input layer
model.add(LSTM(64, return_sequences=True, activation='relu'))
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(len(actions), activation='softmax'))
```

[36]:
```
res = [.7, 1, 0.1]
```

[37]:
```
actions[np.argmax(res)]
```

[37]:
```
'Father'
```

[38]:
```
model.compile(optimizer='Adam', loss='categorical_crossentropy',␣
 ↪metrics=['categorical_accuracy'])
```

[224]:
```
model.fit(X_train, y_train, epochs=950, callbacks=[tb_callback])
```

```
Epoch 1/950
12/12  7s 47ms/step -
categorical_accuracy: 0.0819 - loss: 5.9618
.
categorical_accuracy: 0.9356 - loss: 0.2072
Epoch 199/950
12/12  1s 52ms/step -
categorical_accuracy: 0.7173 - loss: 0.6465
Epoch 200/950
12/12  1s 49ms/step -
```

```
categorical_accuracy: 0.8794 - loss: 0.3431
Epoch 201/950
 7/12  0s 50ms/step -
categorical_accuracy: 0.9222 - loss: 0.2733
```

```
 Layer (type)                 Output Shape              Param #

 lstm_3 (LSTM)                (None, 30, 64)          442,112

 lstm_4 (LSTM)                (None, 30, 128)          98,816

 lstm_5 (LSTM)                (None, 64)               49,408

 dense_3 (Dense)              (None, 64)                4,160

 dense_4 (Dense)              (None, 32)                2,080

 dense_5 (Dense)              (None, 13)                  429


 Total params: 597,005 (2.28 MB)

 Trainable params: 597,005 (2.28 MB)

 Non-trainable params: 0 (0.00 B)
```

8 Making Predictions

[40]:
```
res = model.predict(X_test)
```

```
1/1  0s 433ms/step
```

[41]:
```
actions[np.argmax(res[0])]
```

[41]: 'Thanks'

[42]:
```
actions[np.argmax(y_test[0])]
```

[42]: 'Love'

9 Saving Weights!

[43]:
```
model.save('action.h5')
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')` or `keras.saving.save_model(model,
'my_model.keras')`.
```

[44]:
```
model.load_weights('action.h5')
```

10 Evaluation using Confusion matrix and accuracy

[45]:
```
from sklearn.metrics import multilabel_confusion_matrix, accuracy_score
```

[46]:
```
yhat = model.predict(X_test)
```

```
1/1  0s 37ms/step
```

[47]:
```
ytrue = np.argmax(y_test, axis=1).tolist()
yhat = np.argmax(yhat, axis=1).tolist()
```

[48]:
```
multilabel_confusion_matrix(ytrue, yhat)
```

[48]:
```
array([[[17.,  0.],
        [ 3.,  0.]],

       [[18.,  0.],
        [ 2.,  0.]],

       [[ 7., 13.],
        [ 0.,  0.]],

       [[16.,  0.],
        [ 4.,  0.]],

       [[19.,  0.],
        [ 1.,  0.]],

       [[16.,  0.],
        [ 4.,  0.]],

       [[18.,  0.],
        [ 2.,  0.]],

       [[19.,  0.],
        [ 1.,  0.]],
```

```
       [[12.,  7.],
        [ 1.,  0.]],

       [[19.,  0.],
        [ 1.,  0.]],

       [[19.,  0.],
        [ 1.,  0.]]])
```

[49]:
```
accuracy_score(ytrue, yhat)
```

[49]:
```
0.0
```

11 Testing in Real-Time

[50]:
```
# 1. New detection variables
sequence = []
sentence = []
predictions = []
threshold = 0.5

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.append(keypoints)
        sequence = sequence[-30:]

        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)])
                    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()
```

```
<class 'mediapipe.python.solution_base.SolutionOutputs'>

1/1  0s 431ms/step
Friend
<class 'mediapipe.python.solution_base.SolutionOutputs'>
1/1  0s 20ms/step


Thanks
<class 'mediapipe.python.solution_base.SolutionOutputs'>
1/1  0s 17ms/step
```