1. **# TechVidvan Hand Gesture Recognizer**
2. **# import necessary packages**
3. **# pip install opencv-python**
4. **# pip install mediapipe**
5. **import cv2**
6. **import numpy as np**
7. **import mediapipe as mp**
8. **import tensorflow as tf**
9. **from tensorflow.keras.models import load\_model**
10. **# initialize mediapipe**
11. **mpHands = mp.solutions.hands**
12. **hands = mpHands.Hands(max\_num\_hands=1, min\_detection\_confidence=0.7)**
13. **mpDraw = mp.solutions.drawing\_utils**
14. **Comments:** These lines include comments that provide information about the purpose of the code and instructions for installing necessary packages using **pip**.
15. **Imports:** This section imports required Python libraries and modules:
    * **cv2**: OpenCV library for computer vision tasks.
    * **numpy**: NumPy library for numerical operations.
    * **mediapipe**: A library by Google for hand tracking and gesture recognition.
    * **tensorflow** and **tensorflow.keras.models**: TensorFlow and Keras for machine learning and deep learning.
16. **Initialization of MediaPipe:**
    * **mpHands = mp.solutions.hands**: Creates a MediaPipe Hands object.
    * **hands = mpHands.Hands(max\_num\_hands=1, min\_detection\_confidence=0.7)**: Initializes the Hands object with parameters, specifying that it should detect up to 1 hand with a minimum detection confidence of 0.7.
    * **mpDraw = mp.solutions.drawing\_utils**: Utility functions for drawing landmarks and connections on images.
17. # Load the gesture recognizer model
18. model = load\_model('mp\_hand\_gesture')
19. # Load class names
20. f = open('gesture.names', 'r')
21. classNames = f.read().split('\n')
22. f.close()
23. print(classNames)**Load Gesture Recognizer Model:**
    * **model = load\_model('mp\_hand\_gesture')**: Loads a pre-trained machine learning model for hand gesture recognition using Keras.
    * **f = open('gesture.names', 'r')**: Opens a file containing class names (gestures).
    * **classNames = f.read().split('\n')**: Reads and splits the content of the file into a list of class names.
    * **f.close()**: Closes the file.
    * **print(classNames)**: Prints the loaded class names to the console.

pythonCopy code

# Initialize the webcam cap = cv2.VideoCapture(0)

1. **Initialize Webcam:**
2. **while True:**
3. **# Read each frame from the webcam**
4. **\_, frame = cap.read()**
5. **x, y, c = frame.shape**
6. **# Flip the frame vertically**
7. **frame = cv2.flip(frame, 1)**
8. **framergb = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)**
9. **# Get hand landmark prediction**
10. **result = hands.process(framergb)Video Processing Loop:**
    * A **while** loop is used to continuously process video frames.
    * **\_, frame = cap.read()**: Reads a frame from the webcam.
    * **x, y, c = frame.shape**: Gets the dimensions of the frame.
    * **frame = cv2.flip(frame, 1)**: Flips the frame vertically.
    * **framergb = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)**: Converts the frame to RGB format.
    * **result = hands.process(framergb)**: Processes the RGB frame to get hand landmark predictions using the initialized **Hands** object.

pythonCopy code

className = '' # post process the result if result.multi\_hand\_landmarks: landmarks = [] for handslms in result.multi\_hand\_landmarks: for lm in handslms.landmark: lmx = int(lm.x \* x) lmy = int(lm.y \* y) landmarks.append([lmx, lmy]) # Drawing landmarks on frames mpDraw.draw\_landmarks(frame, handslms, mpHands.HAND\_CONNECTIONS) # Predict gesture prediction = model.predict([landmarks]) classID = np.argmax(prediction) className = classNames[classID] # show the prediction on the frame cv2.putText(frame, className, (10, 50), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (0,0,255), 2, cv2.LINE\_AA) # Show the final output cv2.imshow("Output", frame) if cv2.waitKey(1) == ord('q'): break # release the webcam and destroy all active windows cap.release() cv2.destroyAllWindows()

1. **Hand Landmark Processing and Gesture Prediction:**
   * The code checks if there are multi-hand landmarks in the frame.
   * For each detected hand, it extracts landmarks, draws them on the frame, and predicts the gesture using the loaded model.
   * The predicted gesture is displayed on the frame using OpenCV functions.
   * The video loop continues until the user presses 'q'.
2. **Release Resources:**
   * **cap.release()**: Releases the webcam.
   * **cv2.destroyAllWindows()**: Closes all active OpenCV windows.

This code is a simple example of a hand gesture recognition system using the MediaPipe library for hand tracking and a pre-trained deep learning model for gesture recognition.