CODE EXPLANATION:

* Import necessary packages.
* Initialize models.
* Read frames from a webcam.
* Detect hand key points.
* Recognize hand gestures.

Step 1 – Import necessary packages:

To build this Hand Gesture Recognition project, we’ll need four packages. So first import these.

# import necessary packages for hand gesture recognition project using Python OpenCV

import cv2

import numpy as np

import mediapipe as mp

import tensorflow as tf

from tensorflow.keras.models import load\_model

Step 2 – Initialize models:

**Initialize MediaPipe:**

# initialize mediapipe

mpHands = mp.solutions.hands

hands = mpHands.Hands(max\_num\_hands=1, min\_detection\_confidence=0.7)

mpDraw = mp.solutions.drawing\_utils

* Mp.solution.hands module performs the hand recognition algorithm. So we create the object and store it in mpHands.
* Using mpHands.Hands method we configured the model. The first argument is max\_num\_hands, that means the maximum number of hand will be detected by the model in a single frame. MediaPipe can detect multiple hands in a single frame, but we’ll detect only one hand at a time in this project.
* Mp.solutions.drawing\_utils will draw the detected key points for us so that we don’t have to draw them manually.

**Initialize Tensorflow:**

# Load the gesture recognizer model

model = load\_model('mp\_hand\_gesture')

#Load class names

f = open('gesture.names', 'r')

classNames = f.read().split('\n')

f.close()

print(classNames)

* Using the load\_model function we load the TensorFlow pre-trained model.
* Gesture.names file contains the name of the gesture classes. So first we **open** the file using python’s inbuilt open function and then read the file.
* After that, we read the file using the read() function.

**Output** :

[‘okay’, ‘peace’, ‘thumbs up’, ‘thumbs down’, ‘call me’, ‘stop’, ‘rock’, ‘live long’, ‘fist’, ‘smile’]

The model can recognize 10 different gestures.

Step 3 – Read frames from a webcam:

# Initialize the webcam for Hand Gesture Recognition Python project

cap = cv2.VideoCapture(0)

while True:

# Read each frame from the webcam

\_, frame = cap.read()

x , y, c = frame.shape

# Flip the frame vertically

frame = cv2.flip(frame, 1)

# 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()

* We create a VideoCapture object and pass an argument ‘0’. It is the camera ID of the system. In this case, we have 1 webcam connected with the system. If you have multiple webcams then change the argument according to your camera ID. Otherwise, leave it default.
* The cap.read() function reads each frame from the webcam.
* cv2.flip() function flips the frame.
* cv2.imshow() shows frame on a new openCV window.
* The cv2.waitKey() function keeps the window open until the key ‘q’ is pressed.

Step 4 – Detect hand keypoints:

framergb = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

# Get hand landmark prediction

result = hands.process(framergb)

className = ''

# post process the result

if result.multi\_hand\_landmarks:

landmarks = []

for handslms in result.multi\_hand\_landmarks:

for lm in handslms.landmark:

# print(id, lm)

lmx = int(lm.x \* x)

lmy = int(lm.y \* y)

landmarks.append([lmx, lmy])

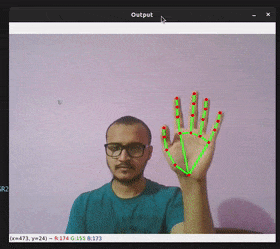
# Drawing landmarks on frames

mpDraw.draw\_landmarks(frame, handslms,

mpHands.HAND\_CONNECTIONS)

* MediaPipe works with RGB images but OpenCV reads images in BGR format. So, using cv2.cvtCOLOR() function we convert the frame to RGB format.
* The process function takes an RGB frame and returns a result class.
* Then we check if any hand is detected or not, using result.multi\_hand\_landmarks method.
* After that, we loop through each detection and store the coordinate on a list called landmarks.
* Here image height (y) and image width(x) are multiplied with the result because the model returns a normalized result. This means each value in the result is between 0 and 1.
* And finally using mpDraw.draw\_landmarks() function we draw all the landmarks in the frame.

**Result:**

[](https://techvidvan.com/tutorials/wp-content/uploads/sites/2/2021/07/landmark-output.gif)

And now, the last and the final step-

Step 5 – Recognize hand gestures:

# Predict gesture in Hand Gesture Recognition project

prediction = model.predict([landmarks])

print(prediction)

classID = np.argmax(prediction)

className = classNames[classID]

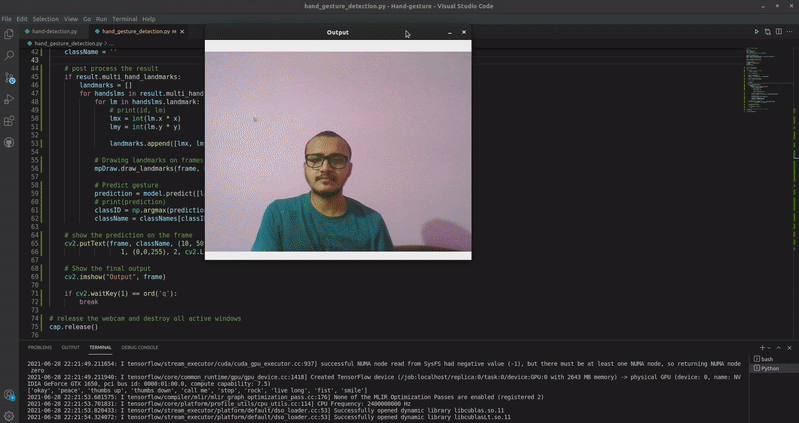
# show the prediction on the frame

cv2.putText(frame, className, (10, 50), cv2.FONT\_HERSHEY\_SIMPLEX,

1, (0,0,255), 2, cv2.LINE\_AA)

* The model.predict() function takes a list of landmarks and returns an array contains 10 prediction classes for each landmark.  
  The output looks like this-  
  [[2.0691623e-18 1.9585415e-27 9.9990010e-01 9.7559416e-05  
  1.6617223e-06 1.0814080e-18 1.1070732e-27 4.4744065e-16 6.6466129e-07 4.9615162e-21]]
* Np.argmax() returns the index of the maximum value in the list.
* After getting the index we can simply take the class name from the classNames list.
* Then using the cv2.putText function we show the detected gesture into the frame.

Hand Gesture Recognition Output

[](https://techvidvan.com/tutorials/wp-content/uploads/sites/2/2021/07/hand-gesture-recognition-output.gif)

Summary

In this Hand Gesture Recognition project, we’ve built a hand gesture recognizer using OpenCV and Python. We’ve used MediaPipe and Tensorflow frameworks for the detection and gesture recognition respectively.