**Code**

!pip install tensorflow tensorflow-gpu opencv-python mediapipe sklearn matplotlib

%pip install tensorflow

import cv2

import numpy as np

import os

from matplotlib import pyplot as plt

import time

import mediapipe as mp

mp\_holistic = mp.solutions.holistic # Holistic model

mp\_drawing = mp.solutions.drawing\_utils # Drawing utilities

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\_landmarks(image, results):

mp\_drawing.draw\_landmarks(image, results.face\_landmarks,

mp\_holistic.FACEMESH\_TESSELATION) # 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

import mediapipe as mp

def draw\_styled\_landmarks(image, results):

# Draw face connections

mp\_drawing.draw\_landmarks(image, results.face\_landmarks,

mp\_holistic.FACEMESH\_TESSELATION,

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),

circle\_radius=2)

mp\_drawing.DrawingSpec(color=(245,66,230), thickness=2,

)

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()

len(results.face\_landmarks.landmark)

mp\_holistic.POSE\_CONNECTIONS

len(results.pose\_landmarks.landmark)

len(results.left\_hand\_landmarks.landmark)

results

draw\_landmarks(frame,results)

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

pose = []

for res in results.pose\_landmarks.landmark:

test = np.array([res.x, res.y, res.z, res.visibility])

pose.append(test)

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)

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)

rh

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])

extract\_keypoints(results).shape

result\_test = extract\_keypoints(results)

result\_test

np.save('0', result\_test)

np.load('0.npy')

# 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','0', '1', '2', '3', '4', '5', '6', '7', '8',

'9','a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v','w','x','y','z'])

# Thirty videos worth of data

no\_sequences = 30

# Videos are going to be 30 frames in length

sequence\_length = 30

# Folder start

start\_folder = 0

for action in actions:

dirmax = np.max(np.array(os.listdir(os.path.join(DATA\_PATH, action))).astype(int))

for sequence in range(1,no\_sequences+1):

try:

os.makedirs(os.path.join(DATA\_PATH, action, str(dirmax+sequence)))

except:

pass

for action in actions:

for sequence in range(no\_sequences):

try:

os.makedirs(os.path.join(DATA\_PATH, action, str(sequence)))

except:

pass

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, '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(500)

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()

pip install scikit-learn

from sklearn.model\_selection import train\_test\_split

from tensorflow.keras.utils import to\_categorical

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

label\_map

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])

np.array(sequences).shape

np.array(labels).shape

X = np.array(sequences)

X.shape

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.5)

y\_test.shape

# UPDATED ONE

import numpy as np

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dense, Dropout, BatchNormalization, Activation

from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping, TensorBoard

from tensorflow.keras.regularizers import l2

# Define the model

model = Sequential()

model.add(LSTM(256, return\_sequences=True, input\_shape=(30, 1662)))

model.add(BatchNormalization())

model.add(Activation('relu'))

model.add(Dropout(0.2))

model.add(LSTM(256, return\_sequences=True))

model.add(BatchNormalization())

model.add(Activation('relu'))

model.add(Dropout(0.2))

model.add(LSTM(128, return\_sequences=False))

model.add(BatchNormalization())

model.add(Activation('relu'))

model.add(Dropout(0.3))

model.add(Dense(128, kernel\_regularizer=l2(0.01)))

model.add(BatchNormalization())

model.add(Activation('relu'))

model.add(Dropout(0.5))

model.add(Dense(actions.shape[0], activation='softmax'))

# Compile the model

model.compile(optimizer='adam', loss='categorical\_crossentropy',

metrics=['categorical\_accuracy'])

# Set up callbacks

lr\_reducer = ReduceLROnPlateau(monitor='val\_loss', factor=0.5, patience=10, min\_lr=1e-6)

early\_stopping = EarlyStopping(monitor='val\_loss', patience=20, restore\_best\_weights=True)

tb\_callback = TensorBoard(log\_dir='Logs')

# Fit the model with validation split

model.fit(X\_train, y\_train, epochs=2000, validation\_split=0.2, callbacks=[tb\_callback,

early\_stopping, lr\_reducer])

model.summary()

res = model.predict(X\_test)

actions[np.argmax(res[4])]

actions[np.argmax(y\_test[4])]

from tensorflow.keras.models import Model

from keras.models import load\_model

model.save("action.h5")

model = load\_model('action.h5')

model.load\_weights('action.h5')

from sklearn.metrics import multilabel\_confusion\_matrix, accuracy\_score

yhat = model.predict(X\_test)

ytrue = np.argmax(y\_test, axis=1).tolist()

yhat = np.argmax(yhat, axis=1).tolist()

multilabel\_confusion\_matrix(ytrue, yhat)

accuracy\_score(ytrue, yhat)

from scipy import stats

colors = [(245,117,16), (117,245,16), (16,117,245)]

import matplotlib.pyplot as plt

#plt.figure(figsize=(18,18))

#plt.imshow(prob\_viz(res, actions, image, colors))

def prob\_viz(res, actions, input\_frame, colors):

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

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.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:]

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()

cap.release()

cv2.destroyAllWindows()