**Libraries:**

import os

import numpy as np

import cv2

import mediapipe as mp

from tensorflow.keras.models import Sequential, load\_model

from tensorflow.keras.layers import LSTM, Dense, Dropout,Bidirectional

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

from tensorflow.keras.utils import to\_categorical

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

from sklearn.metrics import confusion\_matrix, accuracy\_score

import random

import matplotlib.pyplot as plt

from tensorflow.keras.optimizers import Adam # Add this import

# Initialize mediapipe

mp\_holistic = mp.solutions.holistic

mp\_drawing = mp.solutions.drawing\_utils

**# Initialize Mediapipe**

mp\_holistic = mp.solutions.holistic

mp\_drawing = mp.solutions.drawing\_utils

# Function to process images with Mediapipe

def mediapipe\_detection(image, model):

    image = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)

    image.flags.writeable = False

    results = model.process(image)

    image.flags.writeable = True

    image = cv2.cvtColor(image, cv2.COLOR\_RGB2BGR)

    return image, results

# Function to extract keypoints and normalize them

def extract\_keypoints(results, image):

    height, width, \_ = image.shape  # Get image dimensions

    pose = np.array([[res.x \* width, res.y \* height, res.z, res.visibility] for res in results.pose\_landmarks.landmark]) if results.pose\_landmarks else np.zeros((33, 4))

    face = np.array([[res.x \* width, res.y \* height, res.z] for res in results.face\_landmarks.landmark]) if results.face\_landmarks else np.zeros((468, 3))

    lh = np.array([[res.x \* width, res.y \* height, res.z] for res in results.left\_hand\_landmarks.landmark]) if results.left\_hand\_landmarks else np.zeros((21, 3))

    rh = np.array([[res.x \* width, res.y \* height, res.z] for res in results.right\_hand\_landmarks.landmark]) if results.right\_hand\_landmarks else np.zeros((21, 3))

    return np.concatenate([pose.flatten(), face.flatten(), lh.flatten(), rh.flatten()])

# Draw styled landmarks

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

    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)

        )

    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)

        )

    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)

        )

**# Data Augmentation Function**

def augment\_data(image, results):

    # Random flipping

    if random.random() > 0.5:

        image = cv2.flip(image, 1)  # Flip horizontally

    return image, results

**# Define paths and actions**

DATA\_PATH = os.path.join('MP\_Data')

actions = np.array(['Hello', 'Thanks', 'I like it'])  # Define actions

no\_sequences = 20

sequence\_length = 30

# Create necessary directories

for action in actions:

    action\_path = os.path.join(DATA\_PATH, action)

    os.makedirs(action\_path, exist\_ok=True)

**# Open video capture**

cap = cv2.VideoCapture(0)

with mp\_holistic.Holistic(min\_detection\_confidence=0.5, min\_tracking\_confidence=0.5) as holistic:

    for action in actions:

        for sequence in range(no\_sequences):

            for frame\_num in range(sequence\_length):

                ret, frame = cap.read()

                if not ret:

                    print("Failed to grab frame")

                    break

                # Process image and make detections

                image, results = mediapipe\_detection(frame, holistic)

                # Apply data augmentation (if needed)

                image, results = augment\_data(image, results)

                # Draw landmarks

                draw\_styled\_landmarks(image, results)

                # Extract and save keypoints

                keypoints = extract\_keypoints(results, frame)

                npy\_path = os.path.join(DATA\_PATH, action, str(sequence), f'{frame\_num}.npy')

                os.makedirs(os.path.dirname(npy\_path), exist\_ok=True)

                np.save(npy\_path, keypoints)

                if frame\_num == 0:

                    cv2.putText(image, f'STARTING COLLECTION: {action}', (120, 200),

                                cv2.FONT\_HERSHEY\_SIMPLEX, 1, (0, 255, 0), 4, cv2.LINE\_AA)

                    cv2.imshow('OpenCV Feed', image)

                    cv2.waitKey(500)

                else:

                    cv2.putText(image, f'Collecting frames for {action}, Video {sequence}', (15, 12),

                                cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 0, 255), 1, cv2.LINE\_AA)

                    cv2.imshow('OpenCV Feed', image)

                if cv2.waitKey(10) & 0xFF == ord('q'):

                    break

**# Prepare labels and sequences**

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

sequences, labels = [], []

# Load sequences and labels from the data folder

for action in actions:

    action\_path = os.path.join(DATA\_PATH, action)

    for sequence in np.array(os.listdir(action\_path)).astype(int):

        window = []

        for frame\_num in range(sequence\_length):

            # Load keypoint data for each frame

            res = np.load(os.path.join(action\_path, str(sequence), f"{frame\_num}.npy"))

            window.append(res)

        sequences.append(window)

        labels.append(label\_map[action])

X = np.array(sequences)

y = to\_categorical(labels).astype(int)  # One-hot encode the labels

# Split data into training and test sets (stratified splitting to maintain class balance)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.05, stratify=y, random\_state=42)

**# Define model**

model = Sequential()

model.add(LSTM(512, return\_sequences=True, activation='relu', input\_shape=(sequence\_length, 1662)))

model.add(Dropout(0.3))

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

model.add(Dropout(0.3))

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

model.add(Dense(128, activation='relu'))

model.add(Dropout(0.2))  # Added dropout to avoid overfitting

model.add(Dense(64, activation='relu'))

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

**# Compile the model**

model.compile(optimizer=Adam(learning\_rate=0.001), loss='categorical\_crossentropy', metrics=['categorical\_accuracy'])

# Callbacks

log\_dir = os.path.join('Logs')

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

# Early stopping to prevent overfitting

early\_stopping = EarlyStopping(monitor='val\_loss', patience=10, restore\_best\_weights=True)

# Learning rate scheduler to reduce the learning rate after 10 epochs

def lr\_schedule(epoch, lr):

    return lr \* 0.9 if epoch > 10 else lr

lr\_callback = LearningRateScheduler(lr\_schedule)

**Training:**

model.fit(

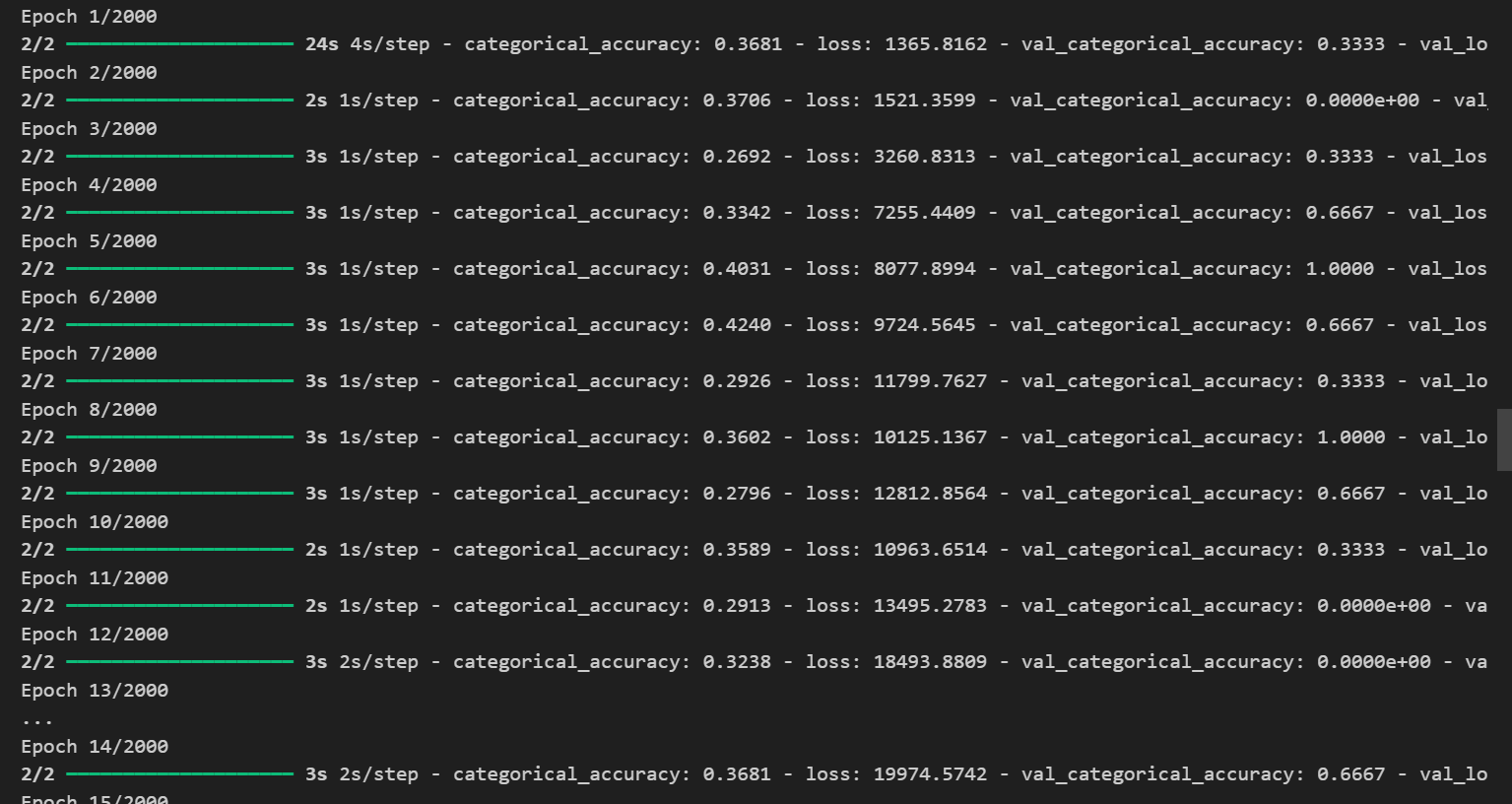
    X\_train, y\_train,

    epochs=2000,

    validation\_data=(X\_test, y\_test),

    callbacks=[tb\_callback, early\_stopping, lr\_callback]

)



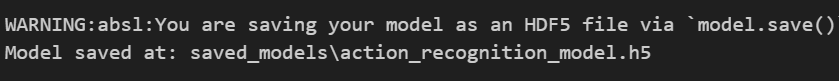
**# Save model**

MODEL\_SAVE\_PATH = os.path.join('saved\_models', 'action\_recognition\_model.h5')

os.makedirs('saved\_models', exist\_ok=True)

model.save(MODEL\_SAVE\_PATH)

print(f"Model saved at: {MODEL\_SAVE\_PATH}")



# Load model (if needed)

 model = load\_model(MODEL\_SAVE\_PATH)

 print("Model loaded successfully!")



**# Test predictions**

yhat = model.predict(X\_test)

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

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

# Confusion Matrix and Accuracy

print("Confusion Matrix:")

print(confusion\_matrix(ytrue, yhat))

print(f"Accuracy: {accuracy\_score(ytrue, yhat)}")

A screenshot of a computer

Description automatically generated

**# Prediction logic and visualization**

threshold = 0.7  # Confidence threshold for predictions

predictions = []

sentence = []

# Start live predictions

cap = cv2.VideoCapture(0)

with mp\_holistic.Holistic(min\_detection\_confidence=0.5, min\_tracking\_confidence=0.5) as holistic:

    while cap.isOpened():

        ret, frame = cap.read()

        if not ret:

            print("Failed to grab frame")

            break

        image, results = mediapipe\_detection(frame, holistic)

        draw\_styled\_landmarks(image, results)

        # Extract keypoints

        keypoints = extract\_keypoints(results, frame)

        predictions.append(keypoints)

        predictions = predictions[-30:]  # Keep only the last 30 frames

        if len(predictions) == 30:  # Only predict once enough frames are collected

            res = model.predict(np.expand\_dims(predictions, axis=0))[0]

            predicted\_action = actions[np.argmax(res)]

            confidence = res[np.argmax(res)]

            if confidence > threshold:

                # Update sentence only if the predicted action is different from the last one

                if len(sentence) == 0 or predicted\_action != sentence[-1]:

                    sentence.append(predicted\_action)

        if len(sentence) > 5:  # Limit sentence length to 5 actions

            sentence = sentence[-5:]

        # Display sentence on the screen

        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)

        cv2.imshow('OpenCV Feed', image)

        # Break the loop if 'q' is pressed

        if cv2.waitKey(10) & 0xFF == ord('q'):

            break

cap.release()

cv2.destroyAllWindows()

