Action Sign Language Interpretation using Deep Learning and MediaPipe

Data Collection Code:

```
import cv2
import numpy as np
import os
import mediapipe as mp

# Extract Keypoints
mp_holistic = mp.solutions.holistic
mp_drawing = mp.solutions.drawing_utils  # Import mediapipe drawing utilities

# Function to draw landmarks and bounding box
def draw_landmarks_and_bbox(frame, results, left_bbox, right_bbox):
    # Draw face landmarks
    if results.face_landmarks:
        mp_drawing.draw_landmarks(
            frame, results.face_landmarks, mp_holistic.FACEMESH_CONTOURS)
    # Draw pose landmarks
    if results.pose_landmarks:
        mp_drawing.draw_landmarks(
            frame, results.pose_landmarks, mp_holistic.POSE_CONNECTIONS)
    # Draw left hand landmarks
    if results.left_hand_landmarks:
        mp_drawing.draw_landmarks(
            frame, results.left_hand_landmarks, mp_holistic.HAND_CONNECTIONS)
    # Draw right hand landmarks
    if results.right_hand_landmarks:
        mp_drawing.draw_landmarks(
            frame, results.right_hand_landmarks, mp_holistic.HAND_CONNECTIONS)
    # Draw bounding box for left hand region
    if left_bbox is not None:
        cv2.rectangle(frame, (int(left_bbox[0]), int(left_bbox[1])), (int(
            left_bbox[0] + left_bbox[2]), int(left_bbox[1] + left_bbox[3])),
(255, 0, 0), 2)
    # Draw bounding box for right hand region
    if right_bbox is not None:
        cv2.rectangle(frame, (int(right_bbox[0]), int(right_bbox[1])), (int(
            right_bbox[0] + right_bbox[2]), int(right_bbox[1] + right_bbox[3])),
(0, 255, 0), 2)


# Function to extract keypoints and bounding box coordinates
def extract_keypoints_and_bbox(results, image_width, image_height):
    keypoints = 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)

    left_hand_landmarks = results.left_hand_landmarks
    right_hand_landmarks = results.right_hand_landmarks

    left_bbox = np.zeros(4)
    right_bbox = np.zeros(4)
```

```
    if left_hand_landmarks:
        min_x, max_x = float('inf'), float('-inf')
        min_y, max_y = float('inf'), float('-inf')
        for landmark in left_hand_landmarks.landmark:
            x, y = landmark.x * image_width, landmark.y * image_height
            min_x = min(min_x, x)
            max_x = max(max_x, x)
            min_y = min(min_y, y)
            max_y = max(max_y, y)
        bbox_width = max_x - min_x
        bbox_height = max_y - min_y
        left_bbox = np.array([min_x, min_y, bbox_width, bbox_height])

    if right_hand_landmarks:
        min_x, max_x = float('inf'), float('-inf')
        min_y, max_y = float('inf'), float('-inf')
        for landmark in right_hand_landmarks.landmark:
            x, y = landmark.x * image_width, landmark.y * image_height
            min_x = min(min_x, x)
            max_x = max(max_x, x)
            min_y = min(min_y, y)
            max_y = max(max_y, y)
        bbox_width = max_x - min_x
        bbox_height = max_y - min_y
        right_bbox = np.array([min_x, min_y, bbox_width, bbox_height])

    # Concatenate all the keypoints and bounding box coordinates
    return np.concatenate([keypoints, face, lh, rh, left_bbox, right_bbox])


# Path for exported data, numpy arrays
DATA_PATH = os.path.join(os.getcwd(), 'Data')

actions = np.array(['none'])

for action in actions:
    action_path = os.path.join(DATA_PATH, action)
    os.makedirs(action_path, exist_ok=True)
    for sequence_number in range(1, 31):
        sequence_path = os.path.join(action_path, f"sequence_{sequence_number}")
        os.makedirs(sequence_path, exist_ok=True)

cap = cv2.VideoCapture(0)
# Set mediapipe model
with mp_holistic.Holistic(min_detection_confidence=0.8,
min_tracking_confidence=0.8) as holistic:
    for action in actions:
        for sequence_number in range(1, 31):
            sequence_keypoints_and_bbox = []
            while len(sequence_keypoints_and_bbox) < 30:  # Collect 30 frames for
each sequence
                ret, frame = cap.read()
                if not ret:
                    break
                # Make detections
                image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
                results = holistic.process(image)
                image_height, image_width, _ = frame.shape
                # Extract keypoints and bounding box coordinates
                keypoints_and_bbox = extract_keypoints_and_bbox(
                    results, image_width, image_height)
                sequence_keypoints_and_bbox.append(keypoints_and_bbox)
```

```
                # Draw landmarks and bounding box on the frame
                draw_landmarks_and_bbox(frame, results, keypoints_and_bbox[-8:-
4], keypoints_and_bbox[-4:])
                cv2.putText(frame, f"Action: {action}, Sequence:
{sequence_number}", (
                    10, 50), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 225), 2,
cv2.LINE_AA)
                cv2.imshow('Data Collection', frame)
                if cv2.waitKey(50) & 0xFF == ord('q'):
                    break
            if len(sequence_keypoints_and_bbox) == 30:  # Save sequence after
collecting 30 frames
                print(
                    f"Collecting data for action: {action}, sequence:
{sequence_number}")
                sequence_path = os.path.join(DATA_PATH, action,
f"sequence_{sequence_number}")
                for frame_num, keypoints_and_bbox in
enumerate(sequence_keypoints_and_bbox):
                    npy_path = os.path.join(sequence_path,
f"frame_{frame_num}.npy")
                    np.save(npy_path, keypoints_and_bbox)
            else:
                break

cap.release()
cv2.destroyAllWindows()
```

Training Code:

```
import cv2
import numpy as np
import os
import mediapipe as mp
from keras.layers import LSTM, Dense, Dropout, BatchNormalization, Reshape,
Bidirectional, TimeDistributed
from keras.optimizers import Adam
from keras.callbacks import EarlyStopping, ReduceLROnPlateau
from keras.regularizers import l2
import numpy as np
from sklearn.model_selection import train_test_split
from keras.utils import to_categorical
from keras.models import Sequential
from keras.losses import categorical_crossentropy

# Path for exported data, numpy arrays
DATA_PATH = os.path.join(os.getcwd(), 'Data')

actions = np.array(['hello', 'love', 'no', 'none', 'thankyou', 'yes'])

# Create a dictionary to map string labels to integer values
label_map = {label: i for i, label in enumerate(actions)}

sequences, labels = [], []
for action in actions:
    for sequence in np.array(os.listdir(os.path.join(DATA_PATH, action))):
        window = []
        for frame_num in range(30):
            res = np.load(os.path.join(DATA_PATH, action, str(sequence),
f"frame_{frame_num}.npy"))
            window.append(res)
```

```
        sequences.append(window)
        labels.append(label_map[action])

X = np.array(sequences)
y = to_categorical(labels)

print("Shape of X before splitting:", X.shape)
print("Number of sequences loaded:", len(sequences))
print("Number of frames per sequence:", len(sequences) // len(labels))

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)

y_test.shape

print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)

# Define and compile your model with the custom optimizer

model = Sequential()
model.add(TimeDistributed(Dense(128), input_shape=(30,1670)))
model.add((LSTM(128, return_sequences=True)))
model.add(Dropout(0.5))
model.add((LSTM(256, return_sequences=True)))
model.add(Dropout(0.5))
model.add((LSTM(128, return_sequences=True)))
model.add(Dropout(0.5))
model.add((LSTM(64)))
model.add(Dropout(0.5))
model.add(Dense(128, activation='relu', kernel_regularizer=l2(0.01)))
model.add(BatchNormalization())
model.add(Dense(len(actions), activation='softmax'))

optimizer = Adam(learning_rate=0.001)

model.compile(optimizer=optimizer, loss='categorical_crossentropy',
              metrics=['accuracy', 'categorical_accuracy'])

# Learning Rate Scheduling and Early Stopping
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2,
                              patience=5, min_lr=0.0001)
early_stop = EarlyStopping(
    monitor='val_loss', patience=10, verbose=1, restore_best_weights=True)

# Now, you can train your model
history = model.fit(X_train, y_train, epochs=50, batch_size=64,
                    validation_data=(X_test, y_test),
                    callbacks=[reduce_lr, early_stop])

# Save the trained model
model.save('action_detection_model.h5')
```

Testing Code:

```
from flask import Flask, render_template, Response, request, redirect, url_for,
session
import cv2
import numpy as np
from keras.models import load_model
import mediapipe as mp
from collections import Counter
import hashlib
import json
import cv2
import mediapipe as mp
import numpy as np
from collections import Counter

app = Flask(__name__)
app.secret_key = 'your_secret_key'

# Load the trained model
model = load_model('action_detection_model.h5')

actions = np.array(['hello', 'love', 'no', 'none', 'thankyou', 'yes'])

# Extract Keypoints
mp_holistic = mp.solutions.holistic
mp_drawing = mp.solutions.drawing_utils  # Import mediapipe drawing utilities

# Function to hash passwords
def hash_password(password):
    return hashlib.sha256(password.encode()).hexdigest()

# Function to draw landmarks and bounding box
def draw_landmarks_and_bbox(frame, results, left_bbox, right_bbox):
    # Draw face landmarks
    if results.face_landmarks:
        mp_drawing.draw_landmarks(
            frame, results.face_landmarks, mp_holistic.FACEMESH_CONTOURS)
    # Draw pose landmarks
    if results.pose_landmarks:
        mp_drawing.draw_landmarks(
            frame, results.pose_landmarks, mp_holistic.POSE_CONNECTIONS)
    # Draw left hand landmarks
    if results.left_hand_landmarks:
        mp_drawing.draw_landmarks(
            frame, results.left_hand_landmarks, mp_holistic.HAND_CONNECTIONS)
    # Draw right hand landmarks
    if results.right_hand_landmarks:
        mp_drawing.draw_landmarks(
            frame, results.right_hand_landmarks, mp_holistic.HAND_CONNECTIONS)
    # Draw bounding box for left hand region
    if left_bbox is not None:
        cv2.rectangle(frame, (int(left_bbox[0]), int(left_bbox[1])), (int(
            left_bbox[0] + left_bbox[2]), int(left_bbox[1] + left_bbox[3])),
(255, 0, 0), 2)
    # Draw bounding box for right hand region
    if right_bbox is not None:
        cv2.rectangle(frame, (int(right_bbox[0]), int(right_bbox[1])), (int(
            right_bbox[0] + right_bbox[2]), int(right_bbox[1] + right_bbox[3])),
(0, 255, 0), 2)

def extract_keypoints_and_bbox(results, image_width, image_height):
    keypoints = 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)

    left_hand_landmarks = results.left_hand_landmarks
    right_hand_landmarks = results.right_hand_landmarks

    left_bbox = np.zeros(4)
    right_bbox = np.zeros(4)

    if left_hand_landmarks:
        min_x, max_x = float('inf'), float('-inf')
        min_y, max_y = float('inf'), float('-inf')
        for landmark in left_hand_landmarks.landmark:
            x, y = landmark.x * image_width, landmark.y * image_height
            min_x = min(min_x, x)
            max_x = max(max_x, x)
            min_y = min(min_y, y)
            max_y = max(max_y, y)
        bbox_width = max_x - min_x
        bbox_height = max_y - min_y
        left_bbox = np.array([min_x, min_y, bbox_width, bbox_height])

    if right_hand_landmarks:
        min_x, max_x = float('inf'), float('-inf')
        min_y, max_y = float('inf'), float('-inf')
        for landmark in right_hand_landmarks.landmark:
            x, y = landmark.x * image_width, landmark.y * image_height
            min_x = min(min_x, x)
            max_x = max(max_x, x)
            min_y = min(min_y, y)
            max_y = max(max_y, y)
        bbox_width = max_x - min_x
        bbox_height = max_y - min_y
        right_bbox = np.array([min_x, min_y, bbox_width, bbox_height])

    return np.concatenate([keypoints, face, lh, rh, left_bbox, right_bbox])

def generate_frames():
    predictions = []
    detected_actions_list = []
    sentence = []
    threshold = 0.8

    cap = cv2.VideoCapture(0)
    cap.set(cv2.CAP_PROP_FRAME_WIDTH, 1280)
    cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 720)

    with mp.solutions.holistic.Holistic(min_detection_confidence=0.8,
min_tracking_confidence=0.8) as holistic:
        sequence_keypoints_and_bbox = []
        while True:
            ret, frame = cap.read()
            if not ret:
                break

            # Resize the frame to 1280x720 using cubic interpolation
```

```
            frame = cv2.resize(frame, (1280, 720), interpolation=cv2.INTER_CUBIC)

            # Make detections
            image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            results = holistic.process(image)
            image_height, image_width, _ = frame.shape

            # Extract keypoints and bounding box coordinates
            keypoints_and_bbox = extract_keypoints_and_bbox(results, image_width,
image_height)
            sequence_keypoints_and_bbox.append(keypoints_and_bbox)

            # Draw landmarks and bounding box on the frame
            draw_landmarks_and_bbox(frame, results, keypoints_and_bbox[-8:-4],
keypoints_and_bbox[-4:])

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

            if len(sequence_keypoints_and_bbox) == 30:
                prediction =
model.predict(np.expand_dims(sequence_keypoints_and_bbox, axis=0))[0]
                predictions.append(np.argmax(prediction))
                action_label = actions[np.argmax(prediction)]

                if np.unique(predictions[-10:])[0] == np.argmax(prediction):
                    if prediction[np.argmax(prediction)] > threshold:
                        if len(sentence) > 0:
                            if action_label != sentence[-1]:
                                sentence.append(action_label)
                        else:
                            sentence.append(action_label)

                if len(sentence) > 5:
                    sentence = sentence[-5:]

            # Draw the detected actions on the frame
            cv2.rectangle(frame, (0, 0), (1280, 50), (245, 117, 16), -1)
            cv2.putText(frame, ' '.join(sentence), (5, 30),
cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2, cv2.LINE_AA)

            # Encode the frame into JPEG format
            ret, buffer = cv2.imencode('.jpg', frame)
            frame = buffer.tobytes()
            yield (b'--frame\r\n'
                   b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n')



# Function to load user data from the file
def load_user_data():
    try:
        with open('user_data.json', 'r') as file:
            data = file.read()
            if data:
                return json.loads(data)
            else:
                return {}  # Return empty dictionary if file is empty
    except FileNotFoundError:
        return {}  # Return empty dictionary if file doesn't exist
    except json.JSONDecodeError:
        return {}  # Return empty dictionary if file has incorrect format
```

```python
# Function to save user data to the file
def save_user_data(users):
    with open('user_data.json', 'w') as file:
        json.dump(users, file)

# Dummy database for storing user credentials (replace with actual database)
users = load_user_data()

@app.route('/', methods=['GET', 'POST'])
def index():
    return redirect(url_for('login'))
    # if request.method == 'POST':
    #     # Handle POST request (e.g., form submission)
    #     # Perform necessary processing
    #     return redirect(url_for('login'))  # Redirect to the login page or any
other appropriate page
    # else:
    #     # Handle GET request (e.g., render the homepage)
    #     return render_template('landing.html')  # Render the homepage template

@app.route('/video_feed')
def video_feed():
    if 'username' not in session:
        return redirect(url_for('login'))
    return Response(generate_frames(), mimetype='multipart/x-mixed-replace;
boundary=frame')

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        # Handle login form submission
        username = request.form['username']
        password = request.form['password']

        # Validate username and password (replace this with your validation
logic)
        if username in users and users[username] == hash_password(password):
            session['username'] = username
            return redirect(url_for('landing'))  # Redirect to landing page after
successful login
        else:
            return render_template('login.html', error='Invalid username or
password')

    # If request method is GET, render the login form
    return render_template('login.html')


@app.route('/signup', methods=['GET', 'POST'])
def signup():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
        if username in users:
            return render_template('signup.html', error='Username already
exists')
        else:
            users[username] = hash_password(password)
            save_user_data(users)  # Save updated user data
            session['username'] = username
            return redirect(url_for('landing'))
    return render_template('signup.html')
```

```python
@app.route('/landing')
def landing():
    if 'username' not in session:
        return redirect(url_for('login'))
    return render_template('landing.html')

@app.route('/logout')
def logout():
    session.pop('username', None)
    return redirect(url_for('login'))

if __name__ == "__main__":
    app.run(debug=True)
```