MACHINE VISION

MCTA 4364

SEMESTER 1 SESSION 2024/2025

CLASS ASSIGNMENT: SIGN LANGUAGE DRIVETHRU

SECTION 1

NO.	NAME	MATRIC NUMBER
1.	CHUI YUAN JIE	2114803
2.	MUHAMMAD ZUHAIL BIN MOHAMED ZUKILAN	2412749
3.	NABIL AMMAR BIN NOOR ADNAN	2026967

TABLE OF CONTENTS

INTRODUCTION **3**
PROBLEM STATEMENT **3**
Bridging the Accessibility Gap 3
PROPOSED SOLUTION **4**
Sign Language Order Drive-Through System 4
OBJECTIVE **4**
KEY FEATURE OF THE SYSTEM **5**
CODING PROCESS **5**
PERFORMANCE EVALUATION **7**
CHALLENGES **8**
IMPROVEMENTS **9**
SOURCE **9**
CODING **24**
DEMONSTRATION **28**

INTRODUCTION

Effective communication is essential for accessibility and inclusivity, yet traditional drive-through systems often fail to accommodate individuals who rely on sign language, creating significant barriers. This project aims to develop a sign language-enabled order system for drive-through services using advanced machine vision techniques. By leveraging deep learning algorithms and computer vision tools, the system interprets sign language gestures in real time, translating them into textual or voice-based outputs to facilitate seamless communication with service providers. The solution focuses on achieving high recognition accuracy across diverse sign languages, real-time processing speeds, and practical deployment, demonstrating the potential of machine vision to enhance accessibility, foster inclusivity, and innovate customer service technology.

PROBLEM STATEMENT

Bridging the Accessibility Gap

Drive-through systems often exclude individuals who use sign language, relying on verbal interaction or interfaces not designed for accessibility. This leads to communication barriers, errors, and delays, leaving sign language users feeling excluded from the convenience these services provide.

The lack of tailored solutions highlights a broader gap in accessible technology, particularly for real-time communication. Integrating machine vision and gesture recognition can bridge this gap, creating an inclusive and seamless experience for sign language users while promoting accessibility in the quick-service industry.

PROPOSED SOLUTION

Sign Language Order Drive-Through System

A machine vision-based approach is suggested as a solution to the problems encountered by sign language users in drive-through settings. This creative solution offers a smooth and inclusive ordering experience by using cutting-edge gesture detection technology to translate sign language in real-time.

The device uses a camera to record motions, then uses machine learning algorithms to convert them into voice or text outputs. These outputs are then integrated into the standard order processing system, ensuring a smooth and efficient interaction. The solution improves overall operational efficiency at drive-throughs and improves the user experience for sign language users by removing communication barriers.

This approach shows how technology may be utilized to create inclusive environments and promote equal access to daily services, marking a significant advancement in accessibility.

OBJECTIVE

In order to solve the communication difficulties that sign language users encounter in drive-through settings, the suggested Sign Language Order Drive-Through System was created with four main goals in mind:

1. Inclusivity: The system seeks to offer the community of people who are hearing or speech challenged a completely accessible solution that will allow them to engage with drive-through services without any difficulties. The solution promotes equal access and improves the user experience by emphasizing inclusion.
2. Precision: Ensuring accurate and dependable sign language gesture recognition is one of the system's main objectives. To reduce errors and correctly understand a variety of motions in real-time, sophisticated machine vision algorithms and reliable training techniques are used.
3. Efficiency: The system is designed to streamline communication, reducing the time required for order placement. By enabling smooth and rapid interaction, the solution enhances both customer satisfaction and operational flow in busy drive-through environments.

4. Scalability: To maximize its impact, the system is built with scalability in mind, ensuring adaptability to various drive-through setups. Whether in fast-food restaurants, pharmacies, or other service industries, the solution can be tailored to meet diverse operational needs.

KEY FEATURE OF THE SYSTEM

The system boasts several key features that enhance its functionality and user experience. **Real-Time Gesture Recognition**, powered by machine vision algorithms, enables instant and intuitive interactions. It also offers **Integration with Existing Systems**, allowing for easy incorporation into current workflows without disruption. The system supports **Customizable Sign Language**, adapting to regional variations for greater inclusivity. Finally, the **User Feedback Mechanism** lets users confirm order details before submission, ensuring accuracy and reducing errors. These features combine to create an efficient, adaptable, and user-friendly system.

CODING PROCESS

1. **Import libraries**

   OpenCV (`cv2`): A library for real-time computer vision tasks

   NumPy (`np`): Provides support for numerical computations and array manipulations.

   OS (`os`): Enables interaction with the operating system, useful for file and directory management.

   Matplotlib (`plt`): A library for creating visualizations and plotting graphs.

   Time (`time`): Used for measuring or manipulating time intervals.

   Mediapipe (`mp`): A framework for building multimodal applications, commonly used for real-time hand, face, and pose detection.

2. **Import holistic model from MediaPipe for landmark detection and OpenCV for video capture**

   The MediaPipe Holistic model allows us to detect pose, hand, and face landmarks. OpenCV enables us to capture video from the webcam and continuously processes each video frame. The landmarks on pose, hands and face were customized to different styles and colors for better distinction. Pressing the ‘q’ key closes the video feed and gracefully exits the program.

3. **Extract landmark keypoints from MediaPipe.**

   Extract pose, face, and hands landmark keypoints and combine all extracted keypoints into a single array.

4. **Set up the directory structure and environment for storing data**

   Create several folders for different hand signs. Since the aim is to create a food order system, folders of 2 drinks and 2 foods as well as ‘yes’ and ‘no’ were created. 40 video sequences were collected for each action, and the video sequences were set to be 30 frames.

5. **Keypoint data collection**

   The data will be collected using the laptop’s cam by loading MediaPipe Holistic model. The number of video samples for each action were 40 videos. The number of frames in each video sequence were 30 frames. Visual cues are included in the code during data collection. The keypoints extracted will be saved in the folders created before.

6. **Processing collected data to prepare it for training a neural network**

   Define actions and map to numerical labels. Prepare sequences and labels. Convert data to Numpy arrays. Split the data into 95% training sets and 5% testing sets.

7. **Training LSTM-based model**

   The model consists of 3 LSTM layers with different neuron sizes, a Dropout to prevent overfitting, 2 dense layers, and an output layer activated with softmax. Tensorboard callback logs the training progress for visualization. ReduceLROnPlateau reduces the learning rate when the validation loss stops improving, helping fine-tune the training. The model is compiled with Adam compiler with categorical crossentropy as loss function. The model trains with 300 epochs. The model is also saved to a file after running the code.

8. **Evaluation of model**

   Obtain the final loss and accuracy on the test data. Utilize TensorBoard to analyze the training process. Implementation of multilabel confusion matrix to test dataset.

9. **Reload model**

Reloading the saved model allows us to use it directly for evaluation, predictions, or deployment without retraining.

10. **Real-time hand sign detection**

    Initialize the Mediapipe holistic model to detect hand landmarks and use the LSTM model to classify gestures into predefined actions. Depending on the detected action, the script updates a list with selected items, removes the last item if "no" is detected, or exits when "yes" is confirmed, signaling order completion. The recognized actions are visually represented with a probability bar, while associated images (e.g., food items) are displayed on-screen. The order list is dynamically saved to a text file, and the system can be exited by pressing 'q'.

11. **Graphical user interface (GUI) as order list**

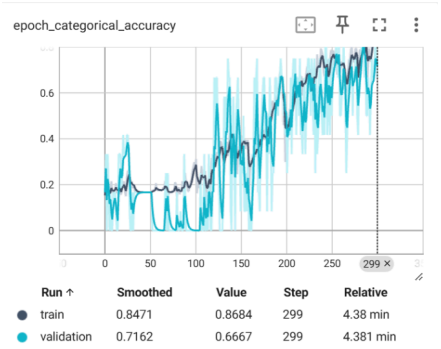
    Creates a simple graphical user interface (GUI) using Tkinter to display the contents of an order list stored in a text file . The GUI updates the displayed order in real-time by reading the file every second.

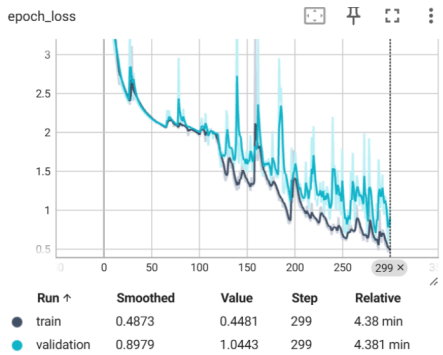
12. **Extra : YOLO object detection as backup**

    Integrates the YOLO object detection model into a real-time video feed to detect specific actions represented by objects or gestures. It uses OpenCV for video processing and Mediapipe for detection. Similar to step 10.

PERFORMANCE EVALUATION

1. LSTM training evaluation





from tensorflow.keras.callbacks import TensorBoard

2. Confusion matrix

```
#switch X_train to X_test and y_train to y_test if you want
from sklearn.metrics import multilabel_confusion_matrix
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)
```

```
array([[[10,  2],
        [ 0,  0]],

       [[11,  1],
        [ 0,  0]],

       [[ 9,  1],
        [ 2,  0]],

       [[10,  0],
        [ 2,  0]],

       [[ 8,  0],
        [ 0,  4]],

       [[ 8,  0],
        [ 0,  4]]], dtype=int64
```

CHALLENGES

Gesture recognition systems face several challenges that can impact their accuracy and performance. One major issue is inconsistent detection, where the data used for training may differ from real-time conditions. Variations in factors such as lighting, background noise, camera angles, or even user movements can introduce inconsistencies in the model's predictions. These discrepancies can affect the system's ability to recognize gestures accurately in real-world scenarios. Additionally, hand gestures that are similar or ambiguous in nature may confuse the model, leading to incorrect interpretations.

Another challenge is incorrect detection, which can arise from a lack of data variety during training. The model may not generalize well to new, unknown data if it is trained on a small dataset that excludes a wide variety of hand motions, backgrounds, lighting conditions, angles, and hand orientations. When the system is exposed to new situations, its performance may suffer due to the training set's lack of diversity.

Last but not least, lagging is an issue, particularly when utilizing computationally intensive Long Short-Term Memory (LSTM) networks. Complex models and large datasets demand a lot of processing power, and the system's reaction time may suffer if specialized

hardware like GPUs is not used. Delays may arise from this, which would impair the gesture recognition system's ability to function in real time.

IMPROVEMENTS

One effective approach is to increase data diversity. The performance of the model heavily depends on the diversity and volume of the training data. To enhance this, data should be collected under a variety of conditions, such as different lighting scenarios, camera angles, and hand orientations. This diversity ensures that the model is better equipped to handle real-world variations. Additionally, techniques like data augmentation can be employed to artificially expand the dataset by introducing slight transformations, such as rotations or scaling, thereby simulating a wider range of input conditions and further improving model robustness.

Regularization techniques are another tactic to enhance generalization and avoid overfitting. More complicated patterns in the data can be captured by making the model more sophisticated, for instance, by including additional LSTM units or layers. However, regularization strategies like Dropout, L2 regularization, and early halting are crucial to preventing overfitting. Early stopping stops the training process when performance on a validation set stops improving, L2 regularization penalizes high weights, and dropout helps keep neurons from co-adapting during training. These methods ensure the model generalizes well to new, unseen data, improving its real-time prediction accuracy.

Incorporating these strategies can enhance data diversity and applying regularization techniques can significantly enhance the robustness and accuracy of gesture recognition systems, making them more reliable for real-world applications.

SOURCE

https://www.youtube.com/watch?v=doDUihpj6ro&t=5046s

https://www.bimsignbank.org/home

CODING

1. Mediapipe and LSTM version (Action Detection)

```
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 CONVERSION RGB
2 BGR
    return image, results

def draw_landmarks(image, results):
    # Draw pose connections
    mp_drawing.draw_landmarks(image, results.pose_landmarks,
mp_holistic.POSE_CONNECTIONS)  # Draw pose connections
    # Draw left hand connections
    mp_drawing.draw_landmarks(image, results.left_hand_landmarks,
mp_holistic.HAND_CONNECTIONS)  # Left hand
    # Draw right hand connections
    mp_drawing.draw_landmarks(image, results.right_hand_landmarks,
mp_holistic.HAND_CONNECTIONS)  # Right hand
    # Draw face landmarks (use mp.solutions.face_mesh instead)
    if results.face_landmarks:
        mp_drawing.draw_landmarks(image, results.face_landmarks,
mp.solutions.face_mesh.FACEMESH_CONTOURS)

def draw_styled_landmarks(image, results):
    # Draw face connections
```

```
    if results.face_landmarks:
        mp_drawing.draw_landmarks(
            image, results.face_landmarks,
mp.solutions.face_mesh.FACEMESH_CONTOURS,
            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),
                              mp_drawing.DrawingSpec(color=(245, 66, 230),
thickness=2, circle_radius=2))

cap = cv2.VideoCapture(0)

# Set the resolution (width x height)
cap.set(cv2.CAP_PROP_FRAME_WIDTH, 1000)  # Set width q
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 600)  # Set height

# 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()
```

```
# step 3

def extract_keypoints(results):

    """

    Extracts keypoint values from MediaPipe results for pose, face, and
hands.

    Returns a single flattened array of keypoints.

    """

    # Extract pose landmarks

    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)
```

```
    # Extract face landmarks
    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)

    # Extract left hand landmarks
    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)

    # Extract right hand landmarks
    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)

    # Concatenate all keypoints into a single array
    return np.concatenate([pose, face, lh, rh])

# Example usage
result_test = extract_keypoints(results)
print("Shape of extracted keypoints:", result_test.shape)
```

```
# step 4

# Path for exported data, numpy arrays

DATA_PATH = os.path.join('MP_Data')



# Actions that we try to detect

actions = np.array(['yes','no','burger','KFC''coke', 'Nescafe'])



# Number of videos worth of data

no_sequences = 40



# Videos are going to be 30 frames in length

sequence_length = 30



# Ensure base directory exists

if not os.path.exists(DATA_PATH):

    os.makedirs(DATA_PATH)



# Create directories for each action and sequence

for action in actions:

    for sequence in range(no_sequences):

        path = os.path.join(DATA_PATH, action, str(sequence))

        try:

            os.makedirs(path)
```

```
            print(f"Created folder: {path}")
        except FileExistsError:
            print(f"Folder already exists: {path}")
```

```
#step 5, collect data, no need to run since alr collected
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(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)
#                 print(results)
```

```
                # 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(2000)
                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()
```

```
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical

actions = ["burger", "yes", "no", "Nescafe","KFC","coke"]

# Map actions to numerical labels
label_map = {label: num for num, label in enumerate(actions)}

# Prepare sequences and labels
sequences, labels = [], []
for action in actions:
    for sequence in range(no_sequences):
        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])

# Convert to numpy arrays
X = np.array(sequences)
y = to_categorical(labels).astype(int)

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.05)

print(f"Training data shape: {X_train.shape}, {y_train.shape}")
print(f"Testing data shape: {X_test.shape}, {y_test.shape}")
```

```
# training LSTM, no need to run this again
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras.callbacks import TensorBoard
from tensorflow.keras.callbacks import ReduceLROnPlateau
from tensorflow.keras.regularizers import l2
import time
```

```
# Create a TensorBoard callback
log_dir = os.path.join('Logs')
if not os.path.exists(log_dir):
    os.makedirs(log_dir)

tb_callback = TensorBoard(log_dir=log_dir)

# Define the model
model = Sequential([
    LSTM(64, return_sequences=True, activation='relu', input_shape=(30,
X_train.shape[2]),kernel_regularizer=l2(0.01)),
    LSTM(128, return_sequences=True,
activation='relu',kernel_regularizer=l2(0.01)),
    LSTM(64, return_sequences=False,
activation='relu',kernel_regularizer=l2(0.01)),
    Dropout(0.3),
    Dense(64, activation='relu'),
    Dense(32, activation='relu'),
    Dense(6, activation='softmax')  # Output layer for classification
])

# Compile the model
model.compile(optimizer='Adam', loss='categorical_crossentropy',
metrics=['categorical_accuracy'])

lr_scheduler = ReduceLROnPlateau(monitor='val_loss', factor=0.5,
patience=3, min_lr=0.0001)

# Train the model
model.fit(
    X_train, y_train,
    epochs=300,  # Use a reasonable number of epochs
    validation_data=(X_test, y_test),
    callbacks=[tb_callback]
)

model.save('test(YN)_action_model.h5')
```

```
model.evaluate(X_test, y_test)
```

```
#reloading model
from tensorflow.keras.models import load_model
model = load_model('test(YN)_action_model.h5')
#You can now use this model to evaluate or predict without retraining.
```

```
#switch X_train to X_test and y_train to y_test if you want
from sklearn.metrics import multilabel_confusion_matrix
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)
```

```
# yes, no control and send output to Gui
import cv2
import mediapipe as mp
import numpy as np
from tensorflow.keras.models import load_model

# Load trained model
model = load_model('test(YN)_action_model.h5')

# Mediapipe initialization
mp_holistic = mp.solutions.holistic
mp_drawing = mp.solutions.drawing_utils

# Actions and variables
actions = ["burger", "yes", "no", "KFC","Nescafe","coke"]
sequence = []
sentence = []
threshold = 0.8

# Visualization helper
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

# Detection loop
cap = cv2.VideoCapture(0)
action_images = {
    "burger": "C:/Users/YUANJIE/OneDrive - International Islamic
University
Malaysia/Documents/machinevision/uma_cv_2024-main/uma_cv_2024-main/hand
sign detection/burgermv.jpg",
    "coke": "C:/Users/YUANJIE/OneDrive - International Islamic University
Malaysia/Documents/machinevision/uma_cv_2024-main/uma_cv_2024-main/hand
sign detection/cokemv.jpg",
    "Nescafe": "C:/Users/YUANJIE/OneDrive - International Islamic
University
Malaysia/Documents/machinevision/uma_cv_2024-main/uma_cv_2024-main/hand
sign detection/nescafemvv.jpg",
    "KFC": "C:/Users/YUANJIE/OneDrive - International Islamic University
Malaysia/Documents/machinevision/uma_cv_2024-main/uma_cv_2024-main/hand
sign detection/kfcmvv.jpg",

}
colors = [(245, 117, 16), (117, 245, 16), (16, 117, 245), (102, 145,
147),(23,200,87),(142,160,178),(231,40,120),(50,245,30)]

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:
            break

        frame = cv2.flip(frame, 1)
        image, results = mediapipe_detection(frame, holistic)

        draw_styled_landmarks(image, results)

        # 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]
            detected_action = actions[np.argmax(res)]
            print(f"Detected action: {detected_action}")

            if detected_action == "yes" and res[np.argmax(res)] >
threshold:
                print("Order completed. Exiting...")
                break
            elif detected_action == "no" and res[np.argmax(res)] >
threshold:
                if sentence:
                    removed = sentence.pop()
                    print(f"Removed: {removed}")
            elif detected_action not in ["yes", "no"] and
res[np.argmax(res)] > threshold:
                if not sentence or detected_action != sentence[-1]:
                    sentence.append(detected_action)
                    print(f"Added: {detected_action}")

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

            # Save order to file
            with open('order_list.txt', 'w') as f:
                for item in sentence:
                    f.write(f"{item}\n")

            # Visualize probabilities
            image = prob_viz(res, actions, image, colors)

            # Display the image corresponding to the top action
            if detected_action in action_images:
                action_image = cv2.imread(action_images[detected_action])
                if action_image is not None:
                    action_image = cv2.resize(action_image, (100, 100))  #
Resize to fit the corner
                    image[50:150, -110:-10] = action_image  # Position the
image at top-right corner
```

```
        # Display current sentence
        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)

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

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

```
import tkinter as tk

def update_order_display():
    try:
        with open('order_list.txt', 'r') as f:
            items = f.readlines()
        order_var.set("\n".join(item.strip() for item in items))
    except FileNotFoundError:
        order_var.set("No items detected yet.")

    root.after(1000, update_order_display)

# Tkinter GUI setup
root = tk.Tk()
root.title("Order List")
root.geometry("300x300")

order_var = tk.StringVar()
label = tk.Label(root, textvariable=order_var, font=("Helvetica", 14),
justify="left")
label.pack(pady=20)

update_order_display()
root.mainloop()
```

2. Roboflow version

```
# ROBOFLOW data
import cv2
from ultralytics import YOLO
import numpy as np
import time
import random

# Load the YOLO model
model = YOLO("projectmv.pt")

# Actions and variables
sentence = []
general_threshold = 0.6
yes_no_threshold = 0.8  # Updated threshold for "yes" and "no" actions

# Action images (update these paths as needed)
action_images = {
    "Burger":
"C:/Users/hp/Downloads/uma_cv_2024-main/uma_cv_2024-main/Projectmv/burgerm
v.jpg",
    "coke":
"C:/Users/hp/Downloads/uma_cv_2024-main/uma_cv_2024-main/Projectmv/cokemv.
jpg",
    "Nescafe":
"C:/Users/hp/Downloads/uma_cv_2024-main/uma_cv_2024-main/Projectmv/nescafe
mvv.jpg",
    "KFC":
"C:/Users/hp/Downloads/uma_cv_2024-main/uma_cv_2024-main/Projectmv/kfcmvv.
jpg",
    "Curry":
"C:/Users/hp/Downloads/uma_cv_2024-main/uma_cv_2024-main/Projectmv/currymv
.jpg",
}

# Function to generate random colors
def generate_colors(n):
    return [(random.randint(0, 255), random.randint(0, 255),
random.randint(0, 255)) for _ in range(n)]
```

```
# Visualization helper function
def prob_viz(res, actions, input_frame, colors):
    output_frame = input_frame.copy()
    for num, prob in enumerate(res):
        color = colors[num] if num < len(colors) else (0, 0, 0)  # Default
to black if colors are exhausted
        cv2.rectangle(output_frame, (0, 60 + num * 40), (int(prob * 100),
90 + num * 40), color, -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

# Open video capture
cap = cv2.VideoCapture(0)

# Set video capture resolution (resize window)
cap.set(cv2.CAP_PROP_FRAME_WIDTH, 640)  # Width 1024px
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)  # Height 768px

while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break

    results = model(frame, stream=True)

    for result in results:
        boxes = result.boxes

        # Check if boxes are empty or not
        if len(boxes) > 0:
            for bbox in boxes:
                x1, y1, x2, y2 = bbox.xyxy[0]
                x1, y1, x2, y2 = int(x1), int(y1), int(x2), int(y2)

                cls_idx = int(bbox.cls[0])
                cls_name = model.names[cls_idx]
                confidence = bbox.conf[0]
```

```
                print(f"Detected: {cls_name} with confidence:
{confidence}")  # Debugging line

                # Draw bounding box and label
                cv2.rectangle(frame, (x1, y1), (x2, y2), (255, 0, 255), 4)
                cv2.putText(frame, cls_name, (x1, y1 - 5),
cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 0, 0), 2)

                # Handle detected actions
                if cls_name in ["yes", "no"] and confidence >
yes_no_threshold:
                    if cls_name == "yes":
                        print("Order completed. Saving and exiting...")
                        with open("order_list.txt", "w") as file:
                            for item in sentence:
                                file.write(f"{item}\n")
                        cap.release()
                        cv2.destroyAllWindows()
                        exit()
                    elif cls_name == "no" and sentence:
                        print("Detected 'no'. Immediately removing the
last item...")
                        removed = sentence.pop()
                        print(f"Removed: {removed}")
                elif cls_name not in ["yes", "no"] and confidence >
general_threshold:
                    if not sentence or cls_name != sentence[-1]:
                        sentence.append(cls_name)

                # Display action images if detected
                if cls_name in action_images:
                    print(f"Displaying image for: {cls_name}")  #
Debugging line
                    action_image = cv2.imread(action_images[cls_name])
                    if action_image is not None:
                        action_image = cv2.resize(action_image, (100,
100))  # Resize to fit the corner
                        frame[50:150, -110:-10] = action_image  # Position
the image at top-right corner
                    else:
```

```
                        print(f"Image for {cls_name} not found!")  #
Debugging line

    # Generate a list of random colors based on sentence length
    colors = generate_colors(len(sentence))

    # Display sentence on the frame
    cv2.rectangle(frame, (0, 0), (1024, 40), (0, 0, 0), -1)  # Adjust
rectangle for sentence display
    cv2.putText(frame, " ".join(sentence), (3, 30),
cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2, cv2.LINE_AA)

    # Visualize the action probabilities (if needed)
    frame = prob_viz([1] * len(sentence), sentence, frame, colors)

    # Show the frame with detection and action images
    cv2.imshow("YOLO Detection", frame)

    if cv2.waitKey(1) & 0xFF == ord("q"):
        break

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

```
import tkinter as tk

def update_order_display():
    try:
        with open('order_list.txt', 'r') as f:
            items = f.readlines()
        order_var.set("\n".join(item.strip() for item in items))
    except FileNotFoundError:
        order_var.set("No items detected yet.")

    root.after(1000, update_order_display)

# Tkinter GUI setup
root = tk.Tk()
root.title("Order List")
root.geometry("300x300")
```

```
order_var = tk.StringVar()
label = tk.Label(root, textvariable=order_var, font=("Helvetica", 14),
justify="left")
label.pack(pady=20)

update_order_display()
root.mainloop()
```

DEMONSTRATION

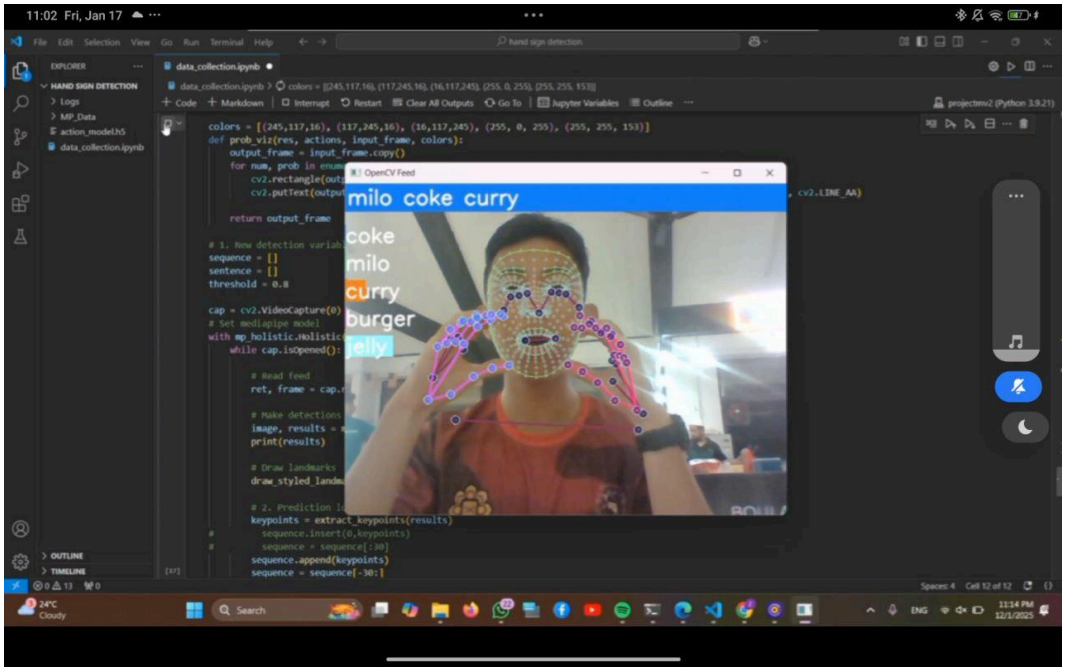


Figure 1: LSTM

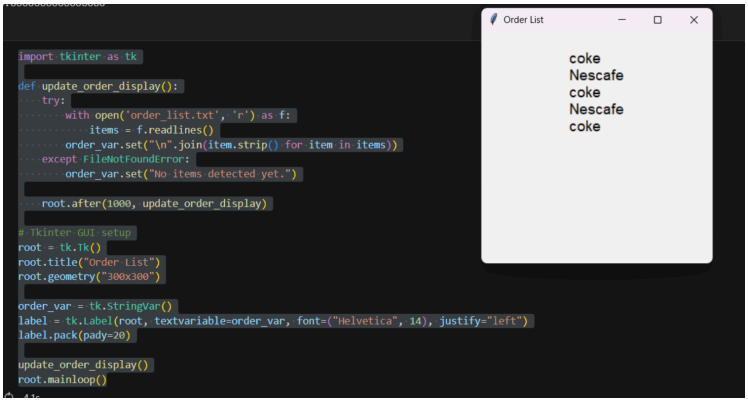


Figure 2: LSTM

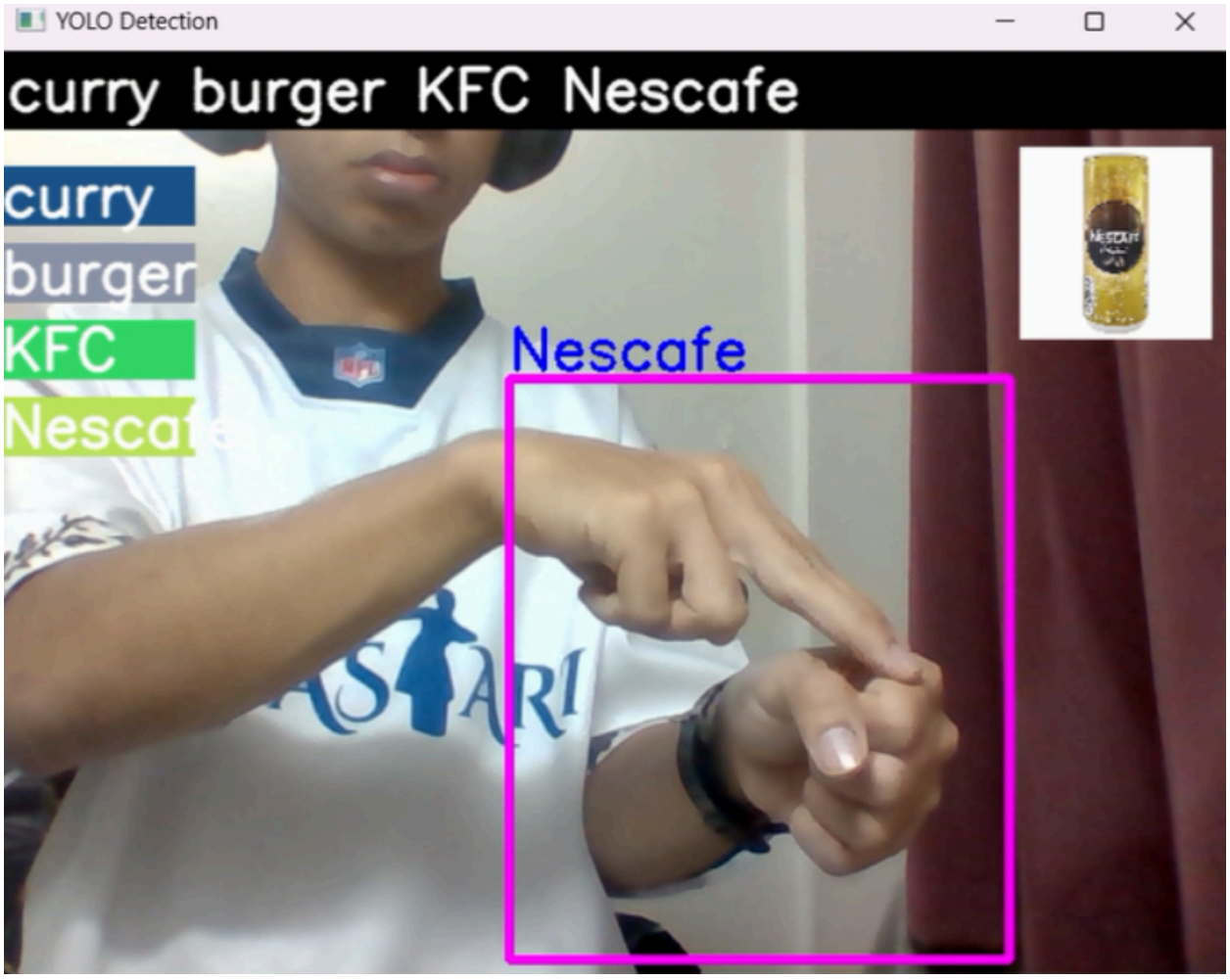


Figure 3: roboflow