SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
RAMAPURAM
FACULTY OF ENGINEERING AND TECHNOLOGY
DEPARTMENT OF INFORMATION TECHNOLOGY

18CSP107L-Minor Project

Final Review

Smart Living with DigiReflect: An IoT Platform for Efficient Task Management

Athira.P-RA2111008020123

Sree Durga Devi.S-RA2111008020135

Anisha.K-RA2111008020151

Dr. K. Danesh

Assistant Professor

ABSTRACT

•DigiReflect: An IoT Platform for Efficient Task Management integrates various productivity tools, including real-time date and time display, a calendar, customizable affirmations, and a note-taking system.

•Built with Python's Tkinter for the GUI and TensorFlow for sign language recognition, it provides an accessible interface for users. The dashboard opens with essential features prominently displayed, allowing easy interaction.

•A key functionality is the integrated sign language detection, which operates in a separate thread, enabling gesture recognition while keeping other features active.

•This application promotes organization and accessibility, catering particularly to individuals who communicate through sign language.

•The seamless interaction between components enhances the user experience, making it a comprehensive tool for daily activities. Overall, it combines productivity and advanced technology to create a user-friendly environment.

PROBLEM IDENTIFICATION

PROBLEM STATEMENT:

The application seeks to create a personal dashboard with real-time sign language detection, note management, and motivational affirmations. It must accurately interpret gestures, allow seamless note editing, and offer an easy-to-navigate interface.

SOLUTION:

A Tkinter-based dashboard integrates a trained TensorFlow model for real-time sign language detection, a note-taking module, and motivational affirmations. Webcam frames are preprocessed for compatibility with the model, while threading ensures responsiveness. The application allows easy note editing, displays a calendar, and updates time and date automatically, providing an accessible, interactive personal dashboard.

DISADVANTAGES

- **Limited Sign Detection Scope**: Recognizes only a specific set of signs, which may not cover all communication needs in sign language.
- **High Resource Usage**: Continuous video processing and machine learning can be resource-intensive, potentially slowing down system performance on lower-spec devices.
- **Dependency on Camera Quality**: Accurate detection requires a good-quality camera and consistent lighting, limiting functionality in low-light conditions or with lower-resolution cameras.
- **Privacy Concerns**: Requires camera access, which could raise privacy issues for some users who may be uncomfortable with continuous video monitoring.
- **Model Accuracy and Generalization**: The model may struggle to accurately interpret varied user gestures, especially with differences in hand positioning or lighting, potentially leading to errors.

MOTIVATION OF THE PROJECT

The motivation behind this project stems from the need to enhance inclusivity, organization, and daily motivation through technology. Sign language detection supports accessibility, making it easier for individuals with hearing or speech impairments to communicate in various environments. Additionally, integrating note-taking, real-time calendar updates, and affirmations within a single dashboard encourages productivity, positivity, and self-reflection. The goal is to build a smart, personal assistant that empowers users to manage daily tasks and stay motivated while embracing the transformative power of artificial intelligence and computer vision for a more inclusive future.

PROPOSED SYSTEM

- **Integrated Dashboard**: Combines sign language recognition, note management, calendar, and affirmations in one interface.
- **Real-Time Sign Language Recognition**: Uses a custom-trained model to detect and classify specific sign language gestures for non-verbal communication.
- **Enhanced Note Module**: Allows users to add, edit, and display notes easily, enabling better task and thought management.
- **Calendar Feature**: Supports event scheduling and tracking for streamlined time management.
- **Affirmations Display**: Centralized, positive affirmations designed to promote well-being and motivation throughout the day.
- **Inclusive Design**: Aims to support a broader range of users, including those needing non-verbal, gesture-based interaction.

ADVANTAGES

- **Enhanced Accessibility**: Facilitates non-verbal communication for users with speech or hearing impairments through real-time sign language detection.
- **Multi-Functionality**: Combines multiple features (notes, calendar, affirmations) in a single dashboard, reducing the need for multiple apps.
- **Improved Productivity**: Allows users to manage tasks, notes, and schedules conveniently, fostering better organization.
- **Positive Reinforcement**: Displays motivational affirmations to support a positive mindset throughout the day.
- **User-Friendly**: Simple, intuitive design with easy-to-navigate features, catering to diverse user needs.
- **Real-Time Interaction**: Instantaneous gesture recognition ensures prompt response, making the system feel responsive and reliable.

SYSTEM REQUIREMENTS

SOFTWARER EQUIREMENTS:

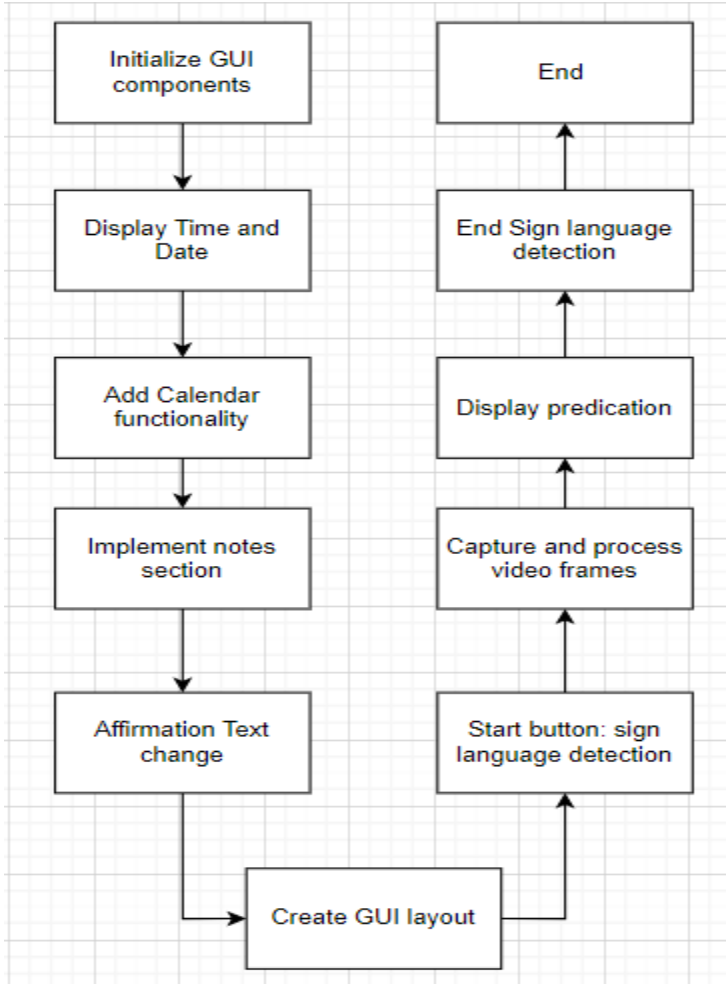
- **Programming Language:**Python3.x
- **Libraries:**
  - OpenCV
  - Tkcalendar
  - Tkinter
  - TensorFlow
  - Matplotlib
  - NumPy
- **Operating System:**
  - Windows10 or higher
  - Raspberry pi OS
- **Integrated Development Environment(IDE):**
  - Thonny
  - Visual Studio Code(VSCode)

SYSTEM REQUIREMENTS

HARDWARE REQUIREMENTS:

- **Web cam:** least 720p resolution for clear image capture.
- **Computer Specifications: Processor-** Intel i3 or above (or equivalent AMD),**RAM-** Minimum 4 GB (8 GB or higher recommended for TensorFlow and OpenCV tasks), **Storage-** At least 500 MB of free space for the project files, libraries, and trained model.
- **Microcontroller:** Raspberry pi pico

SYSTEM ARCHITECTURE



Full Modules presentation with algorithms

Dashboard Module:

- Displays the date, time, affirmations, a note-taking section, and allows users to view a calendar.
- It also integrates the sign language detection functionality, controlled via buttons.

Algorithm:

- **Initialize GUI**: Create a ***Tkinter*** window and set its properties (title, size, background).
- **Display Current Time**: Use ***datetime*** to fetch and update the current time every second.
- **Show Calendar**: Implement a calendar using ***tkcalendar***, allowing users to select dates.
- **Affirmations**: Create a list of affirmations and cycle through them when clicked.
- **Notes Section**: Allow users to add, edit, and display notes in a text area.

Full Modules presentation with algorithms

Sign language detector module:

- **Initialize Webcam**: Use ***OpenCV*** to access the webcam.
- **Frame Capture**: Capture video frames continuously.
- **Preprocess Frames**: Convert frames to grayscale.
- Resize images to fit the model input.
- **Load Trained Model**: Use ***TensorFlow*** to load the pre-trained sign language detection model.
- **Make Predictions**: Predict the sign language gesture from the processed frame and display the result on the frame.

SAMPLE CODE

Thonny - C:\Users\durgas\OneDrive\Documents\sign_language_detection.py @ 169 : 1

File Edit View Run Tools Help

sign_language_detection.py

```python
import os
import tkinter as tk
from tkinter import simpledialog
from tkcalendar import Calendar
from datetime import datetime
import cv2
import numpy as np
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing.image import img_to_array
from sklearn.preprocessing import LabelEncoder
import threading

# Main window setup for Smart Mirror
root = tk.Tk()
root.title("Smart Mirror")
root.geometry("800x600")
root.configure(bg='black')
root.attributes("-fullscreen", True)  # Start in full screen mode

# Font settings
large_font = ("Helvetica", 40, "bold")
medium_font = ("Helvetica", 20)
small_font = ("Helvetica", 12)

# Global list to store notes
notes = []

# Load the trained model and LabelEncoder for sign language detection
```

Shell

```
1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
>>>
```

SAMPLE CODE

Thonny - C:\Users\durgas\OneDrive\Documents\sign_language_detection.py @ 169 : 1

File Edit View Run Tools Help

sign_language_detection.py

```
27
28  # Load the trained model and LabelEncoder for sign language detection
29  model = load_model('gesture_model.keras')
30  gestures = ['yes', 'please', 'help']
31  label_encoder = LabelEncoder()
32  label_encoder.fit(gestures)
33
34  # Function to update time and date
35  def update_time_date():
36      now = datetime.now()
37      current_time = now.strftime("%H:%M:%S")
38      current_date = now.strftime("%B %d, %Y")
39      current_day = now.strftime("%A")
40
41      time_label.config(text=current_time)
42      date_label.config(text=current_date)
43      day_label.config(text=current_day)
44
45      root.after(1000, update_time_date)  # Update every second
46
47  # Function to toggle calendar visibility
48  def show_calendar():
49      cal_win = tk.Toplevel(root)
50      cal_win.title("Calendar")
51      cal_win.geometry("300x300")
52      cal = Calendar(cal_win, selectmode="day")
53      cal.pack(pady=20)
```

Shell

```
1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
>>>
```

SAMPLE CODE

Thonny - C:\Users\durgas\OneDrive\Documents\sign_language_detection.py @ 169 : 1

File Edit View Run Tools Help

sign_language_detection.py

```
# Function to add a new note
def add_note():
    note = simpledialog.askstring("New Note", "Enter your note:")
    if note:
        notes.append(note)
        update_notes_display()

# Function to update the note display area
def update_notes_display():
    notes_display.delete("1.0", tk.END)
    for i, note in enumerate(notes, 1):
        notes_display.insert(tk.END, f"{i}. {note}\n")

# Function to change affirmation text on click
def change_affirmation():
    affirmations = ["You are amazing!", "Stay positive!", "Believe in yourself!", "Keep going!", "You got this!"]
    current_affirmation = affirmation_label.cget("text")
    new_affirmation = affirmations[(affirmations.index(current_affirmation) + 1) % len(affirmations)]
    affirmation_label.config(text=new_affirmation)

# Webcam control functions for gesture recognition
cap = None  # Webcam video capture object

def preprocess_frame(frame, target_size=(64, 64)):
    # Resize the frame to match the input size of the model
    frame_resized = cv2.resize(frame, target_size)
    # Normalize and convert to array
```

Shell

```
1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
>>>
```

SAMPLE CODE

Thonny - C:\Users\durgas\OneDrive\Documents\sign_language_detection.py @ 169 : 1

File Edit View Run Tools Help

sign_language_detection.py

```
    # Expand dimensions to make it compatible with the model input shape
    frame_array = np.expand_dims(frame_array, axis=0)
    return frame_array

def start_webcam():
    global cap
    cap = cv2.VideoCapture(0)  # Open the webcam

    def detect_gestures():
        while cap.isOpened():
            ret, frame = cap.read()
            if not ret:
                break
            # Preprocess the frame for prediction
            preprocessed_frame = preprocess_frame(frame)
            # Make a prediction on the preprocessed frame
            predictions = model.predict(preprocessed_frame)
            predicted_index = np.argmax(predictions, axis=1)[0]
            predicted_gesture = label_encoder.inverse_transform([predicted_index])[0]
            # Display the predicted gesture on the frame
            cv2.putText(frame, f"Gesture: {predicted_gesture}", (10, 30),
                        cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
            # Show the frame with the predicted gesture
            cv2.imshow('Real-time Gesture Prediction', frame)
            if cv2.waitKey(1) & 0xFF == ord('q'):
                break
```

Shell

```
1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
>>>
```

SAMPLE CODE

Thonny - C:\Users\durgas\OneDrive\Documents\sign_language_detection.py @ 169 : 2

File Edit View Run Tools Help

sign_language_detection.py

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

    # Start gesture recognition in a separate thread to avoid blocking the UI
    threading.Thread(target=detect_gestures, daemon=True).start()

def stop_webcam():
    global cap
    if cap and cap.isOpened():
        cap.release()
        cv2.destroyAllWindows()

# UI Elements
# Time, Date, and Day Labels (Top left)
time_label = tk.Label(root, text="", font=large_font, fg="white", bg="black")
time_label.grid(row=0, column=0, sticky="nw", padx=20, pady=10)

date_label = tk.Label(root, text="", font=medium_font, fg="white", bg="black")
date_label.grid(row=1, column=0, sticky="nw", padx=20)

day_label = tk.Label(root, text="", font=medium_font, fg="white", bg="black")
day_label.grid(row=2, column=0, sticky="nw", padx=20)

# Calendar Button (Below Date and Day)
calendar_icon = tk.Button(root, text="", font=large_font, command=show_calendar, bg="black", fg="white", bd=0)
calendar_icon.grid(row=3, column=0, sticky="nw", padx=20, pady=20)

```

Shell

```
1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
>>>
```

SAMPLE CODE

Thonny - C:\Users\durgas\OneDrive\Documents\sign_language_detection.py @ 163 : 1

File Edit View Run Tools Help

sign_language_detection.py *

```
affirmation_label.grid(row=4, column=1, sticky="n", pady=20)
affirmation_label.bind("<Button-1>", lambda e: change_affirmation())  # Change affirmation on click

notes_label = tk.Label(root, text="Notes", font=medium_font, fg="white", bg="black")
notes_label.grid(row=5, column=2, sticky="ne", padx=20, pady=10)

notes_display = tk.Text(root, height=6, width=30, font=small_font, bg="black", fg="white", bd=0)
notes_display.grid(row=6, column=2, sticky="ne", padx=20)

add_note_button = tk.Button(root, text="Add Note", command=add_note, font=small_font, bg="white", fg="black")
add_note_button.grid(row=7, column=2, sticky="ne", padx=20)

start_button = tk.Button(root, text="Start Gesture Detection", font=small_font, bg="green", fg="white", command=start_webcam)
start_button.grid(row=8, column=1, pady=20)

stop_button = tk.Button(root, text="Stop Gesture Detection", font=small_font, bg="red", fg="white", command=stop_webcam)
stop_button.grid(row=9, column=1, pady=20)

root.grid_rowconfigure(4, weight=1)
root.grid_columnconfigure(1, weight=1)


update_time_date()

root.mainloop()

root.mainloop()

```

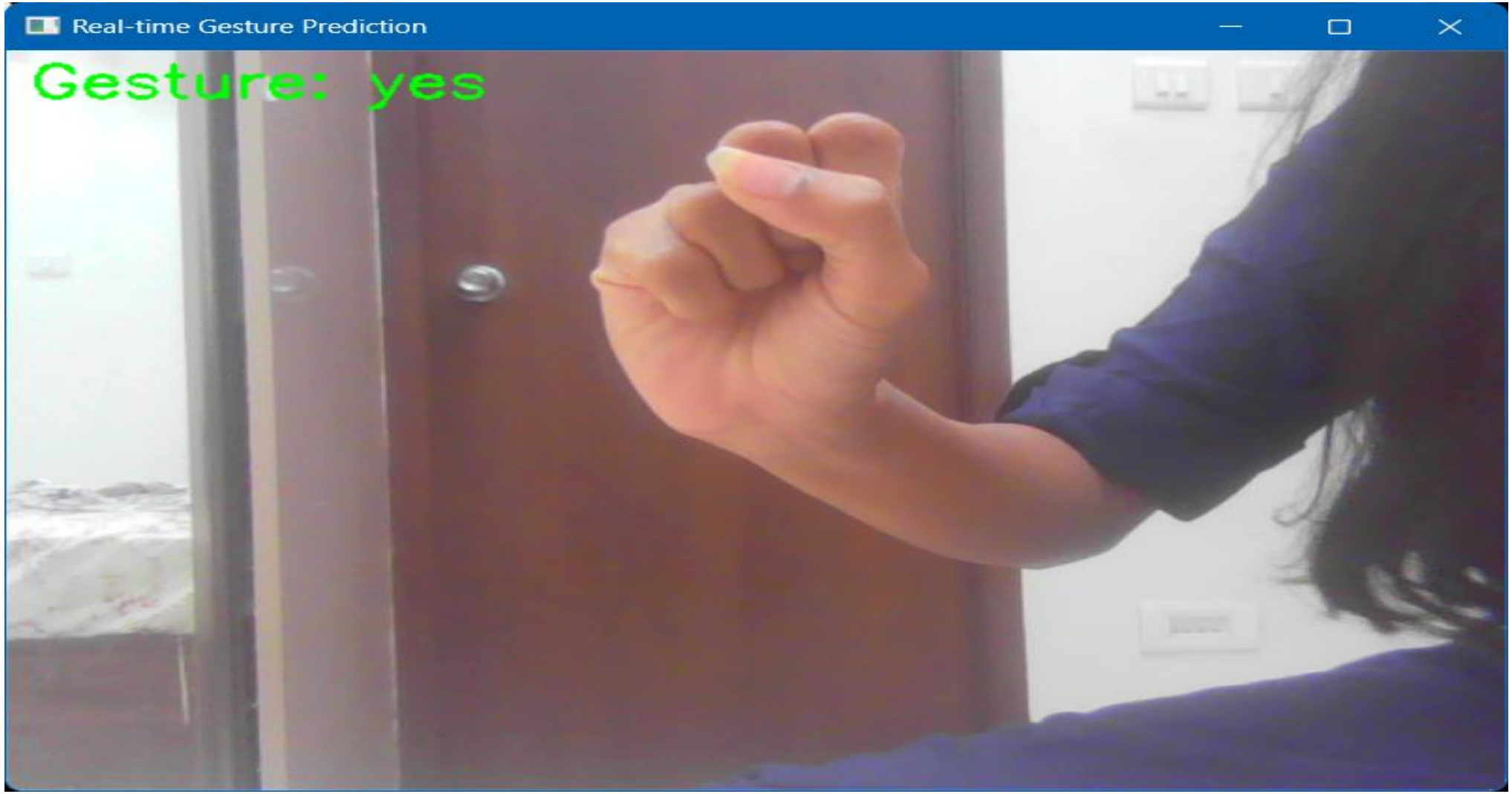
Shell

```
1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
>>>
```

DASHBOARD INTERFACE

CALENDAR POP UP

SMART MIRROR DISPLAY



SMART MIRROR DISPLAY

CONCLUSION

In conclusion, the application successfully integrates multiple productivity tools to enhance user experience. The real-time display of date and time, interactive calendar, affirmations, and note-taking functionality offers users a comprehensive platform for managing daily tasks.

The inclusion of sign language detection enables better communication for users who rely on gestures, making the application more inclusive.

Overall, this project not only demonstrates the practical application of Python and machine learning technologies but also addresses the importance of accessibility in everyday tools.

FUTURE ENHANCEMENT

To improve the functionality and usability of the application, the following enhancements are recommended:

- **Enhanced Sign Language Detection**: Expand the model to recognize more gestures and improve accuracy by incorporating a larger dataset.
- **User Profiles**: Allow users to create profiles to save personalized settings, notes, and affirmations.
- **Data Export**: Implement functionality to export notes and affirmations to a file for backup.
- **Mobile Application**: Develop a mobile version of the dashboard for easier access on various devices.
- **Integration with Other APIs**: Integrate with external APIs (e.g., weather, task management) to provide more comprehensive information.

THANK YOU