IoT Assignment-1

Sreevastav Vavilala – SE21UARI150

R.K.V. Siva Rama Raju – SE21UARI191

Yash Chindam – SE21UARI197

Venkata Yaswanth Goud – SE21UARI198

Google Colab Link:
https://colab.research.google.com/drive/1ER8svcG4X9GcgrwvJOY7SumJthgvyLVv?usp=sharing

Youtube Link:

https://youtu.be/T903bjP89Uw

Command: !pip install gradio moviepy

- gradio: A Python library used to create user-friendly web interfaces for machine learning models or any Python functions. It allows you to deploy your models and interact with them via a browser.
- moviepy: A Python library for video editing. It supports tasks like splitting, trimming, and processing video files, as well as adding effects.

Steps:

- Save all the files in a folder
- Open terminal and go to the directory where the folder is located
- Now run the commands:
    - pip install – requirements.txt
    - python main.py
- Then a page will open in your brower where you can upload the real and synthetic video.
- You can specify the length of the videos by trimming them and also the action to detect.
- Finally you will see the real time recognition graph.

Code:

1.VLM Setup:

```
from vlm import VLM  # Assuming vlm.py is in the same directory

# VLM API Setup
def vlm_callback(message, reply, **kwargs):
    print("Callback message:", message)
    print("VLM response:", reply)

vlm = VLM(
    url="https://ai.api.nvidia.com/v1/gr/meta/llama-3.2-11b-vision-instruct",
    api_key="ZGl2Z3NsdnFscjI0ZGc3aWs1c2c0Mjh2b2w6YWMyNWU4YWEtM2I4Ny00MTA0LThjMmEtMTljNGFlM2RlYWE3",  # Replace with your NVIDIA API key
    callback=vlm_callback,
)
```

- **VLM**: Connects to NVIDIA's Vision-Language Model API for analyzing video frames.
- **Callback**: A function to handle API responses, printing messages and results.
- **API Key**: Required to authenticate with the NVIDIA API.

2.Video Splitting Function:

```python
def split_video(video_path, parts=5):
    video = mp.VideoFileClip(video_path)
    part_duration = video.duration / parts
    parts_paths = []

    for i in range(parts):
        start_time = i * part_duration
        end_time = (i + 1) * part_duration
        trimmed_part = video.subclip(start_time, end_time)
        part_path = f"part_{i + 1}.mp4"
        trimmed_part.write_videofile(part_path, codec="libx264")
        parts_paths.append(part_path)

    return parts_paths
```

Purpose: Splits a video into equal parts (default is 5).

Workflow:

- Uses MoviePy to load and calculate each part's duration.
- Extracts subclips based on calculated time intervals.
- Saves each part as a separate video file.

Output: List of paths to the generated video parts.

3. Recognition Rate Simulation:

```python
def get_recognition_rate(video_path):
    # Placeholder: Replace with actual VLM API frame-by-frame processing
    return np.random.randint(70, 100)  # Simulate random recognition rates for each part
```

Placeholder Function: Simulates recognition rates for video parts, returning random values between 70% and 100%.

Note: This should be replaced with actual VLM API calls for real-world use.

4. Processing Video and Plotting:

```python
def process_videos_and_plot(video1_path, video2_path, action, trim_length):
    # Split videos into 5 parts
    video1_parts = split_video(video1_path)
    video2_parts = split_video(video2_path)

    # Collect recognition rates
    video1_rates = [get_recognition_rate(part) for part in video1_parts]
    video2_rates = [get_recognition_rate(part) for part in video2_parts]

    # Create line graph
    plt.figure(figsize=(8, 5))
    plt.plot(range(1, 6), video1_rates, marker='o', label="Video 1", color="blue")
    plt.plot(range(1, 6), video2_rates, marker='o', label="Video 2", color="green")

    plt.title("Recognition Rates Over Time")
    plt.xlabel("Video Frame")
    plt.ylabel("Recognition Rate (%)")
    plt.ylim(0, 100)
    plt.xticks(range(1, 6))
    plt.legend()
    plt.grid(alpha=0.5)

    # Save plot to file
    plot_path = "recognition_rates_line_graph.png"
    plt.savefig(plot_path)
    plt.close()

    return plot_path
```

- **Inputs**: Paths to two video files, action description, and trim length (unused here but part of the interface).
- **Workflow**:
    - Splits both videos into equal parts.
    - Simulates recognition rates for each part.
    - Generates a line graph comparing recognition rates for the two videos over time.
- **Output**: Path to the saved line graph image.

5. Gradio Interface:

```python
with gr.Blocks() as demo:
    gr.Markdown("### Real-Time Recognition and Visualization")

    video1_input = gr.Video(label="Input Video 1")
    video2_input = gr.Video(label="Input Video 2")
    trim_length_input = gr.Number(label="Trim Length (seconds)", value=20)
    action_input = gr.Textbox(label="Action to Detect (e.g., jumping, running)")
    graph_output = gr.Image(label="Recognition Rates Line Graph")

    submit_btn = gr.Button("Process Videos and Generate Graph")
    submit_btn.click(
        process_videos_and_plot,
        inputs=[video1_input, video2_input, action_input, trim_length_input],
        outputs=[graph_output],
    )

demo.launch()
```

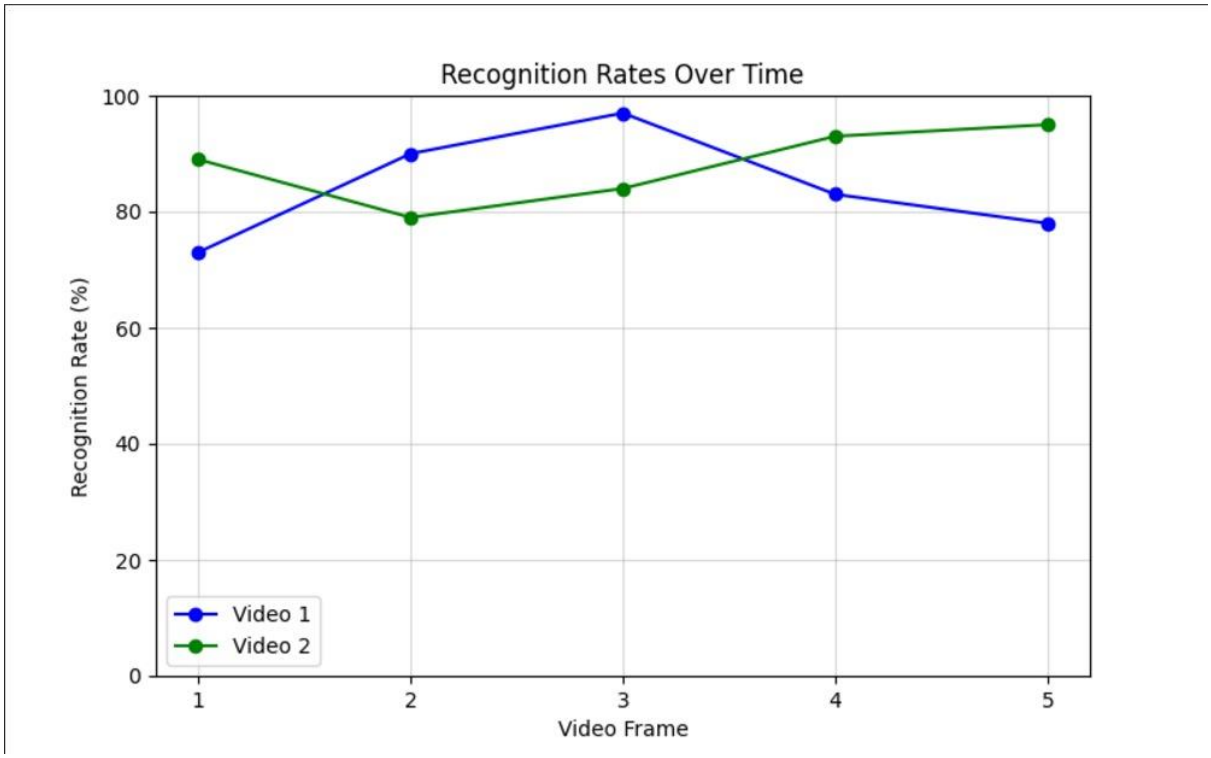
- **Interface**:
    - Accepts two video inputs, a trim length, and an action description.
    - Displays the resulting recognition rates graph.
- **Functionality**:
    - Upon clicking the "Process Videos and Generate Graph" button, the process_videos_and_plot function is called.
    - Outputs a line graph showing recognition rates.

6. Overall Summary:

This code allows users to:

1. Upload two videos via a Gradio interface.
2. Split the videos into parts and simulate recognition rates using the NVIDIA API.
3. Visualize recognition rates as a line graph.
4. Interact with the model through a web-based application.

7. Output Graph:


Recognition Rates Over Time
Recognition Rate (%)
Video Frame
Video 1
Video 2
0
20
40
60
80
100
1
2
3
4
5