Documentation for the work done on Action Detection using NVIDEA VLM

Team:

Varun Kumar Konjeti - ID:SE21UARI186

Susmitha Reddy Vajrala – ID:SE21UARI180

Koushik Dupaguntla – ID:SE21UARI162

Venkata Srinivas Gurram – ID:SE21UARI190

Objective

The primary objective of this project is to develop an interface and evaluation system for comparing real and synthetic human action videos, leveraging NVIDIA's Video Language Models (VLM) to assess the effectiveness of recognition tasks on these datasets

Introduction

Human activity recognition plays a critical role in various applications, ranging from surveillance and healthcare to sports analysis and gaming. With the growing capabilities of synthetic data generation, it has become increasingly important to compare the effectiveness of action detection systems on both real and synthetic data. This project explores the capabilities of NVIDIA’s Vision-Language Models (VLMs) for recognizing human activities in synthetic and real videos.

The primary motivation behind this project is to evaluate the consistency and reliability of action detection algorithms when applied to synthetic and real datasets. With advancements in AI-based video generation, synthetic data has become a cost-effective and scalable alternative to real data for training machine learning models. However, the performance gap between detecting actions in

synthetic and real videos needs to be quantified for ensuring robust model generalization.

The goal of this project is to provide a comprehensive tool that enables researchers to compare human activity detection rates across real and synthetic video data. Using NVIDIA's NEVA API, the project delivers a user-friendly interface for uploading, trimming, and analyzing videos. By leveraging Gradio, the application offers an intuitive platform for users to assess the performance of activity detection algorithms. The insights gained from this project could guide the development of more efficient and consistent action recognition models, improving their applicability across various domains.

a. Setup Instructions

1. **Prerequisites**:
    a. Install Python (version 3.8+ recommended).

    b. Install the required libraries by running:
        pip install -r requirements.txt

    c. Ensure ffmpeg is installed for video processing. Install it using:
        sudo apt install ffmpeg # For Linux
        brew install ffmpeg # For macOS

2. Setup Instructions

a. Clone the repository from GitHub:

        git clone https://github.com/VarunKonjeti12/NVIDIA_Human_Action_Detection_VLM_Workflow.git

        cd NVIDIA_Human_Action_Detection_VLM_Workflow

    b. Launch the application:

python app.py

There is also another method (Most preferred):

This is using the Colab notebook for running the code
The Colab link:
https://colab.research.google.com/drive/1W5yyBGnnLgYzThOLI48WNkpjHB9icMdi?usp=sharing

Just run the cells in the colab, you will be navigated to the gradio interface. (Prefer this method as it is faster and work very well

b. How to use:

1. Launching the Interface:
    a. After running app.py, a local Gradio interface will open in your browser.
2. Uploading Videos:
    a. Upload one synthetic video (Video A) and one real video (Video B).
    b. Supported formats: .mp4, .avi, .mov.
3. Specify an Activity:
    a. Enter the activity you want to detect (e.g., walking, jumping).
4. Optional Trimming:
    a. Enter a trim length (in seconds). If left blank, videos will be trimmed to the duration of the shorter video.
5. Results:
    a. Click “Submit” to get the success rates for both videos.
    b. View results in percentage format.

c. Outputs to Expect

6. Sample Output:
    a. Example:
        i. Synthetic Video: "Walking Success Rate: 72.50%"
        ii. Real Video: "Walking Success Rate: 85.00%"
7. Error Scenarios:
    a. No Trim Length or Invalid Length:
        i. Error: "Trim length must be greater than 0 seconds."
    b. Invalid Video Format:

i. Error: "Failed to extract frames from one or both videos."

Main Code:

```
import requests
import base64
from moviepy.video.io.VideoFileClip import VideoFileClip
from PIL import Image
from io import BytesIO
import gradio as gr

# NVIDIA NEVA API setup
API_ENDPOINT = "https://ai.api.nvidia.com/v1/vlm/nvidia/neva-22b"
AUTH_TOKEN = "nvapi-FB-eOnyYZkMMeB_LxxjcVQjxmEDG5v8P93hrGn-
HqsYrbjbciGNlestq5DJJK2Tj"  # Use securely for production

def capture_frames video_file  total_frames=16
    """Capture equally spaced frames from the provided video."""
    try:
        video = VideoFileClip(video_file)
        video_length = video

        if video_length == 0
            raise ValueError "The video has a duration of zero seconds. Check the file."

        # Extracting frames at evenly spaced intervals
        extracted_frames =
            video.get_frame(i * video_length / total_frames) for i in range(total_frames)
        ]
        return [Image.fromarray(frame_data) for frame_data in extracted_frames]

    except Exception as err
        print f"Error in capture_frames: {err}"
        return []

def convert_image_to_base64  image
    """Convert an image to a Base64-encoded string."""
    try:
        buffer = BytesIO()
        image         buffer  format="PNG"    # Using PNG format as required
        encoded_data = base64 b64encode buffer getvalue      decode
```

```
        return encoded_data
    except Exception as err
        print f"Error in convert_image_to_base64: {err}"
        return ""

def query_neva_api image_base64  activity
    """Send a request to NVIDIA NEVA API to detect the specified activity in the image."""
    if not image_base64
        print "Image data is missing, skipping API call."
        return False

    headers =
        "Authorization"  f"Bearer {AUTH_TOKEN}"
        "Accept"  "application/json"
    }
    payload =
        "messages"
            {
                "role": "user",
                "content"  f'Do you observe the action "{activity}" in this image? <img
src="{image_base64}" />'
            }
        ],
        "max_tokens"  512   # Reduced token count for efficiency
        "temperature"  0.7   # Adjusted temperature for more consistent responses
        "top_p"  1.0         # Increased top_p for better result coverage
        "stream": False,
    }

    try:
        response = requests post  API_ENDPOINT  headers=headers  json=payload  timeout=30   #
Added timeout

        if response status_code == 200
            response_data = response json
            if 'choices' in response_data and response_data 'choices'
                response_content = response_data['choices'][0].get('message', {}).get('content', "")
                return "yes" in response_content.lower()
            else:
                print "No valid choices returned in the response."
        else:
            print f"API call failed with status code: {response status_code}"
```

```
    except requests exceptions Timeout
        print "The request timed out."
    except Exception as err
        print f"Error in query_neva_api: {err}"
    return False

def compute_detection_rate images activity
    """Calculating the percentage of frames where the activity is detected."""
    try:
        positive_detections = 0

        for image in images:
            image_b64 = convert_image_to_base64 image
            if query_neva_api image_b64 activity
                positive_detections += 1

        return positive_detections / len images * 100 if images else 0

    except ZeroDivisionError
        print "No frames available for detection rate calculation."
        return 0

def trim_videos video1_path video2_path trim_length=None
    """Trim two videos to the same duration based on user-provided or shorter video duration."""
    try:
        clip1 = VideoFileClip(video1_path)
        clip2 = VideoFileClip(video2_path)
        min_duration = min clip1 clip2

        # Check if the user-provided trim length is valid
        if trim_length is not None
            if trim_length <= 0:
                return "Error: Trim length must be greater than 0 seconds." None None
            trim_duration = min trim_length min_duration
        else:
            trim_duration = min_duration

        return None clip1 0 trim_duration clip2 0 trim_duration
    except Exception as err
        print f"Error in trim_videos: {err}"
        return "Error: Failed to trim videos. Please check your inputs." None None
```

```
def analyze_videos video_a_path  video_b_path  activity  trim_length
    """Analyze two videos for the specified activity and return success rates."""
    trim_result  trimmed_a  trimmed_b = trim_videos video_a_path  video_b_path  trim_length
    if isinstance trim_result  str    # This means an error occurred
        return trim_result  None

    frames_a = capture_frames trimmed_a
    frames_b = capture_frames trimmed_b

    if not frames_a or not frames_b
        return "Error: Failed to extract frames from one or both videos."  None

    success_rate_a = compute_detection_rate frames_a  activity
    success_rate_b = compute_detection_rate frames_b  activity

    return
        f"Video A '{activity}' Success Rate: {success_rate_a:.2f}%"
        f"Video B '{activity}' Success Rate: {success_rate_b:.2f}%"
    )

# Gradio interface for action detection comparison
ui = gr.Interface(
    fn=analyze_videos
    inputs=
        gr       label="Synthetic Video (Input A)"  file_types= ".mp4"  ".avi"  ".mov"
        gr       label="Real Video (Input B)"  file_types= ".mp4"  ".avi"  ".mov"
        gr            label="Specify the Activity (e.g., walking, jumping)"
        gr            label="Trim Length (seconds, optional)"  value=None
    ],
    outputs=
        gr            label="Analysis for Synthetic Video"
        gr            label="Analysis for Real Video"
    ],
    title="Action Detection via NVIDIA NEVA"
    description="Compare action detection success rates between a synthetic and a real video.
Optionally, specify a trim length."
)

if __name__ == "__main__"
    ui.launch()
```

Code Explanation:

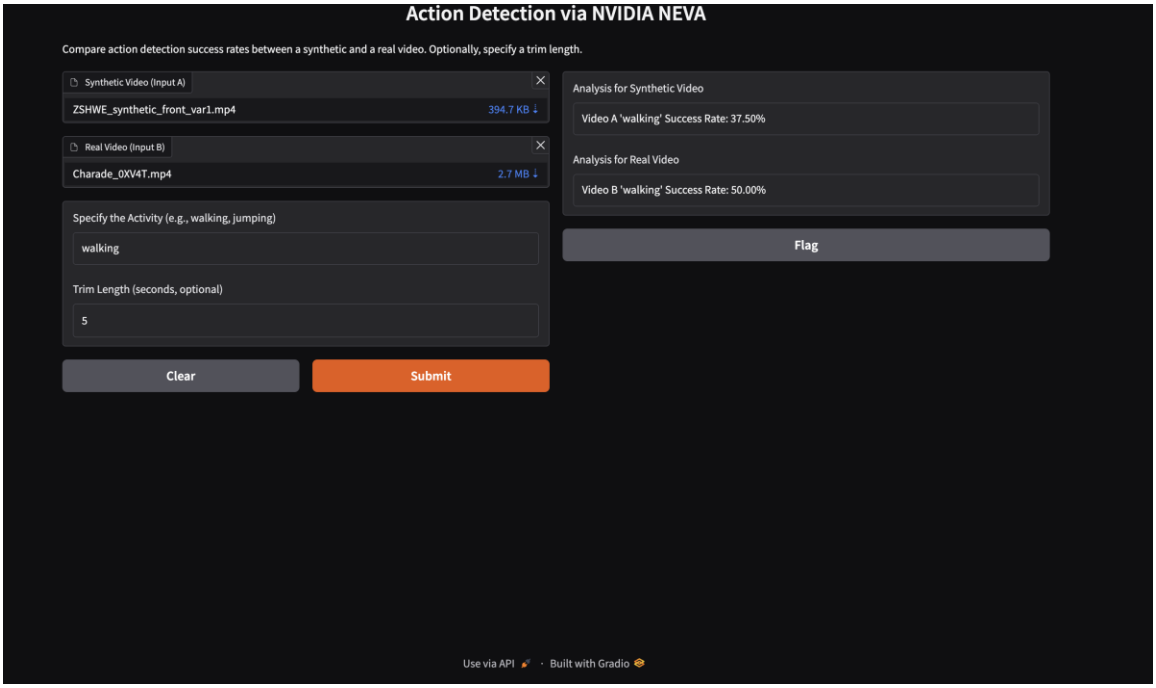
This Python script implements a Gradio-based interface for comparing human activity recognition rates in synthetic and real videos using NVIDIA's NEVA API. The program first sets up the NEVA API endpoint and authentication token, allowing secure communication with NVIDIA's cloud service. It includes functions to process videos and extract frames. The capture_frames function extracts a specified number of evenly spaced frames from a video, converting them to images using the MoviePy library. These frames are then converted into Base64-encoded strings with the convert_image_to_base64 function, preparing them for transmission to the NEVA API.

The query_neva_api function sends the encoded images to the API, querying whether a specific activity (e.g., "walking" or "jumping") is observed in the frames. The function uses a payload that includes the image and activity as input, and it processes the API response to determine if the activity was detected. Detection results are aggregated in the compute_detection_rate function, which calculates the percentage of frames where the activity is identified.

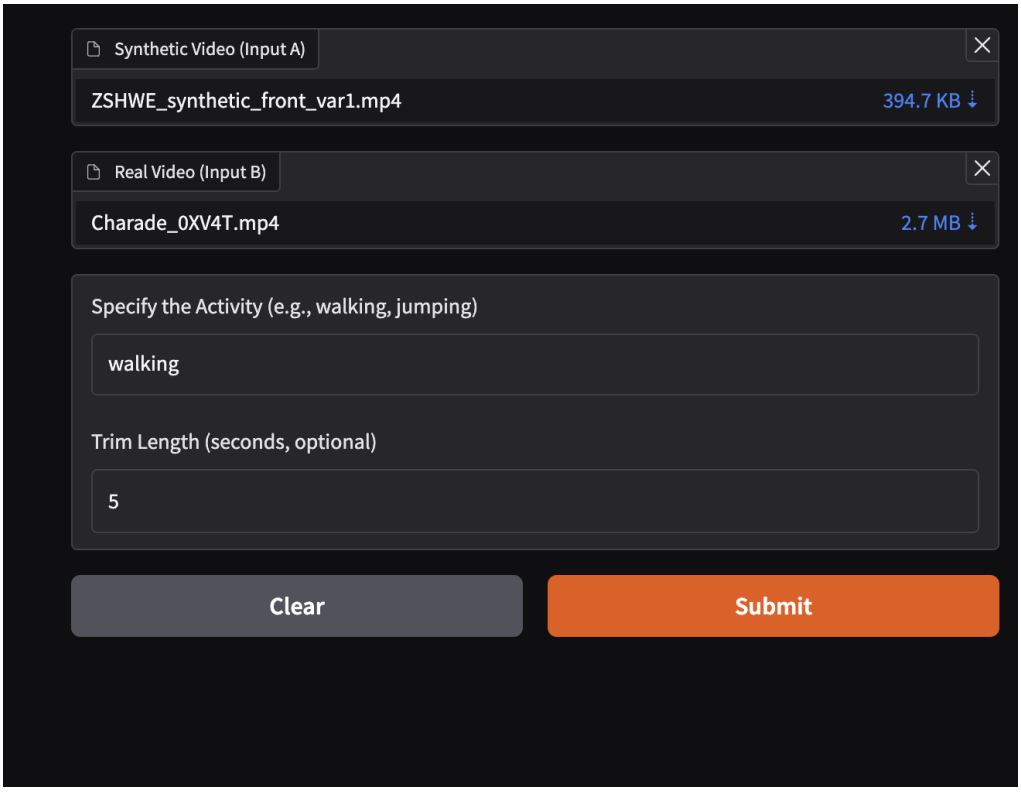
For preprocessing, the script provides the trim_videos function, which trims two videos to a common duration based on the shorter video's length or a user-specified trim length. The analyze_videos function integrates all these steps, trimming the videos, extracting frames, and computing activity recognition success rates for both the synthetic and real videos.

Finally, the Gradio interface allows users to upload two videos, specify an activity, and optionally set a trim length. The interface outputs the recognition success rates for the two videos, helping users compare activity detection performance. The script uses error handling for issues like invalid video files or API failures and ensures a user-friendly experience by logging errors and providing appropriate messages. The Gradio app is launched with a customizable interface when the script is executed.

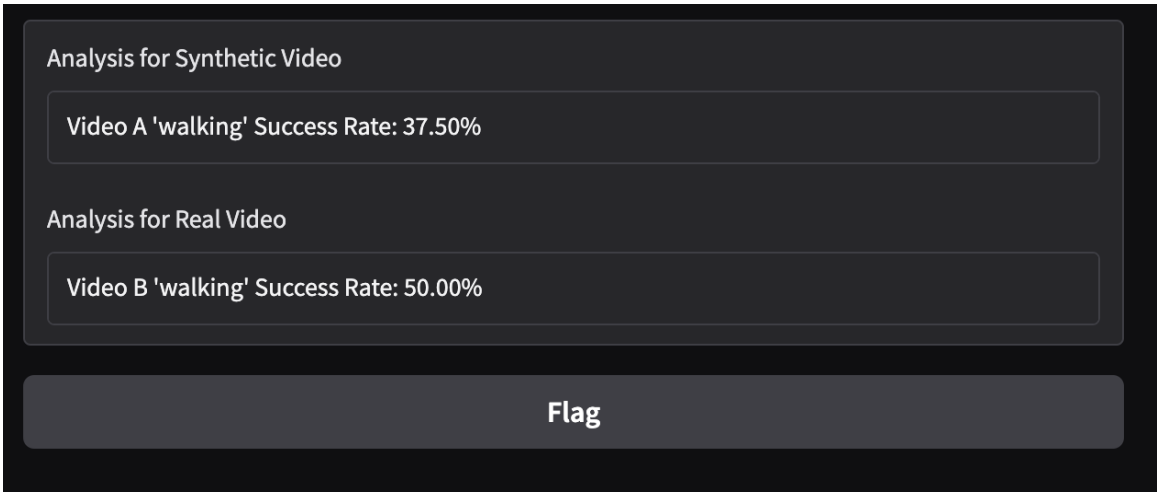
Results and Outputs



Inputs:



Outputs:


Analysis for Synthetic Video
Video A 'walking' Success Rate: 37.50%
Analysis for Real Video
Video B 'walking' Success Rate: 50.00%
Flag