VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

School of Computer Science and Engineering

(SCOPE)

MTech-Computer Science with Business Analytics

EXPERIMENT – 4

Spatio- Temporal segmentation task

CSE4076 - Image and Video Analytics

BY:

Vishali Sharma

21MIA1066

Submitted to:

Dr. Saranyaraj D

OBJECTIVE

- **Load Video**:
  Efficiently load and preprocess the specified video file for subsequent analysis.
- **Extract Frames**:
  Extract individual frames from the video at defined intervals to support further processing.
- **Spatio-Temporal Segmentation**:
  Utilize segmentation techniques, including color thresholding and edge detection, to differentiate and track objects across frames, isolating the foreground from the background.
- **Detect Scene Cuts**:
  Identify hard cuts through pixel and histogram comparisons, and detect soft cuts by analyzing intensity variations between consecutive frames.
- **Highlight Scene Cuts**:
  Mark frames where scene cuts are detected and compile a summary of identified scene boundaries for easy reference.
- **Visualize Results**:
  Display frames with detected scene cuts and showcase segmentation results for selected frames to enhance visual analysis.

PROBLEM STATEMENT:

Description of the Problem:
In the age of digital media, analyzing video content efficiently is crucial for various applications, including video editing, content moderation, and automated video summaries. Existing methods often struggle to accurately identify scene transitions and track objects within a video. This project aims to address these challenges by developing a robust system that processes video files to extract frames, segment foreground and background, and detect scene cuts. The focus will be on implementing both hard cut and soft cut detection techniques to ensure a comprehensive understanding of the video structure.

Expected Output:
The final output of the system will include the following elements:

1. A set of extracted frames from the input video.
2. Segmented images showing identified foreground objects and background using color thresholding and edge detection techniques.
3. A list of detected scene cuts, categorized into hard cuts and soft cuts, along with their respective frame indices.
4. Visual representations of the frames with marked scene cuts, demonstrating where transitions occur.

5. A summary document that includes details about detected scene boundaries, highlighting key points of interest for further analysis or editing.

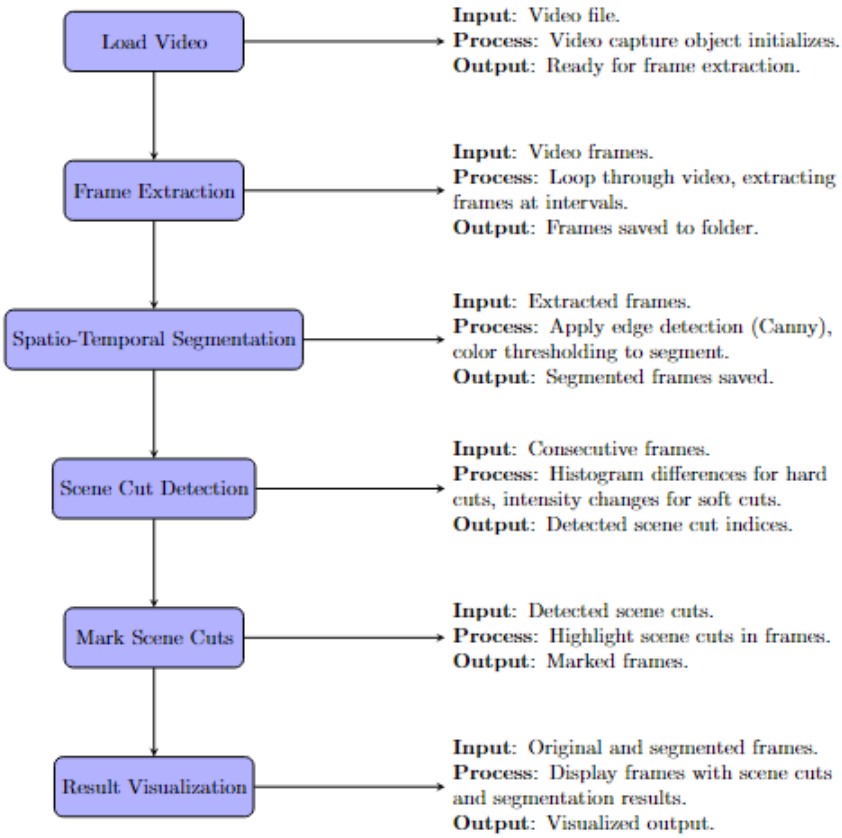
METHODOLOGY:

Algorithm

1. **Load Video:**
    - Open the video file.
    - Initialize the video capture object.
2. **Frame Extraction:**
    - Create an empty list to store frames.
    - Loop through the video and extract frames at defined intervals.
    - Save each frame to a specified output folder.
3. **Spatio-Temporal Segmentation:**
    - For each extracted frame:
        - Apply edge detection (e.g., Canny) to detect edges.
        - Use color thresholding to segment foreground objects from the background.
        - Save the segmented frames to output folders.
4. **Scene Cut Detection:**
    - Calculate histograms for consecutive frames.
    - Detect hard cuts using histogram differences.
    - Analyze frame-to-frame intensity changes to identify soft cuts.
    - Store the indices of detected scene cuts.
5. **Mark Scene Cuts:**
    - Highlight the frames where scene cuts are detected.
    - Save or display these marked frames.
6. **Result Visualization:**
    - Display the original frames with marked scene cuts.
    - Show segmentation results for selected frames.
7. **Summary Output:**

- Create a summary document detailing the detected scene cuts, including the total number of hard and soft cuts, along with their frame indices.

Block – Diagram



PYTHON – IMPLEMENTATION

1. **Load Video**:

Load the provided video file.

```
5]: import cv2

def load_video(video_path):
    """Loads the video and returns the video capture object."""
    cap = cv2.VideoCapture(video_path)
    if not cap.isOpened():
        raise Exception(f"Error: Unable to open video file {video_path}")
    return cap
video_path = '33871-398473585_small.mp4'  # Your video file path
cap = load_video(video_path)
```

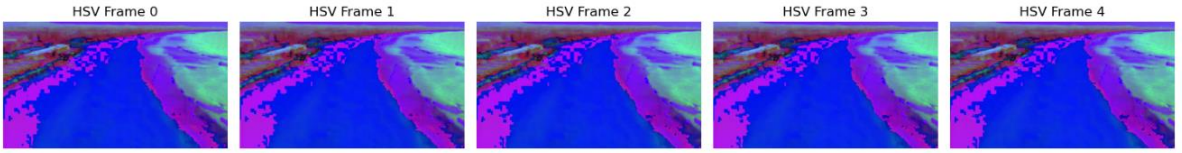
2. **Frame Extraction**:

```python
def extract_frames(cap, output_folder):
    """Extracts frames from the video and saves them to the output folder."""
    os.makedirs(output_folder, exist_ok=True)  # Create the output directory if it doesn't exist
    frames_output = []  # List to hold extracted frames
    frame_index = 0

    while True:
        ret, frame = cap.read()
        if not ret:
            break  # Exit loop if no more frames

        # Save the frame to the output folder
        frame_filename = os.path.join(output_folder, f"frame_{frame_index:04d}.jpg")
        cv2.imwrite(frame_filename, frame)

        frames_output.append(frame)  # Store the frame for further processing
        frame_index += 1

    cap.release()  # Release the video capture
    print(f"Total frames extracted: {frame_index}")

    # Plot the first 5 frames, if available
    plot_first_five_frames(frames_output)

    return frames_output

def plot_first_five_frames(frames_output):
    """Plots the first five frames from the extracted frames."""
    num_frames_to_plot = min(5, len(frames_output))  # Ensure we don't exceed available frames

    plt.figure(figsize=(15, 5))
    for i in range(num_frames_to_plot):
        plt.subplot(1, 5, i + 1)
        plt.imshow(cv2.cvtColor(frames_output[i], cv2.COLOR_BGR2RGB))
```

```python
def plot_first_five_frames(frames_output):
    """Plots the first five frames from the extracted frames."""
    num_frames_to_plot = min(5, len(frames_output))  # Ensure we don't exceed available frames

    plt.figure(figsize=(15, 5))
    for i in range(num_frames_to_plot):
        plt.subplot(1, 5, i + 1)
        plt.imshow(cv2.cvtColor(frames_output[i], cv2.COLOR_BGR2RGB))
        plt.axis('off')
        plt.title(f'Frame {i}')

    plt.show()  # Show the plotted frames

# Example usage
# Assuming `cap` is your video capture object
output_folder = 'frames_output'  # Folder to save extracted frames
frames_output = extract_frames(cap, output_folder)
```

```
Total frames extracted: 628
```

```
    # Create a figure to hold the subplots
    plt.figure(figsize=(15, 5))

    # Loop through each frame to plot
    for i in range(num_frames_to_display):
        plt.subplot(1, num_frames_to_display, i + 1)  # Create a subplot for each frame
        plt.imshow(hsv_frames[i], cmap='hsv')  # Display the HSV frame with the 'hsv' colormap
        plt.axis('off')  # Hide the axis
        plt.title(f'HSV Frame {i}')  # Title for each frame

    plt.tight_layout()  # Adjust the layout for better spacing
    plt.show()  # Show the plotted frames

# Example usage
# Assuming `hsv_frames` is your list of HSV frames
display_hsv_frames(hsv_frames)
```

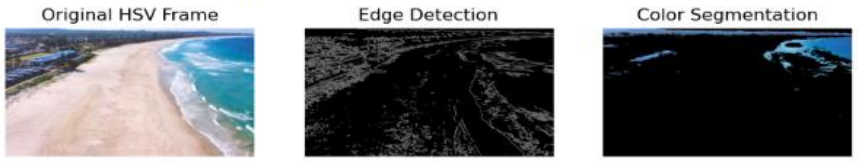


3. **Spatio-Temporal Segmentation**:

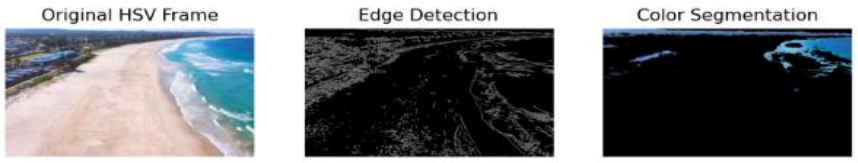
- Perform segmentation on each frame using a technique like color thresholding or edge detection.

```
def perform_segmentation_hsv(input_folder, edge_output_folder, color_output_folder):
    # Create output directories for edge detection and color segmentation
    os.makedirs(edge_output_folder, exist_ok=True)
    os.makedirs(color_output_folder, exist_ok=True)

    # Iterate through all extracted HSV frames
    frame_files = [f for f in os.listdir(input_folder) if f.endswith('.png')]  # Assuming your HSV fr

    for idx, frame_file in enumerate(frame_files):
        frame_path = os.path.join(input_folder, frame_file)

        # Read the HSV frame image
        hsv_frame = cv2.imread(frame_path)

        # --- Edge Detection (Canny) ---
        # Convert the HSV frame to grayscale for edge detection
        # This step is necessary as Canny expects a single channel (grayscale) image
        gray_frame = cv2.cvtColor(hsv_frame, cv2.COLOR_HSV2BGR)  # Convert HSV to BGR to grayscale
        gray_frame = cv2.cvtColor(gray_frame, cv2.COLOR_BGR2GRAY)
        edges = cv2.Canny(gray_frame, 100, 200)

        # Save the edge-detected image
        edge_filename = os.path.join(edge_output_folder, f"edge_{frame_file}")
        cv2.imwrite(edge_filename, edges)

        # --- Color Thresholding ---
        # Define lower and upper bounds for color segmentation in HSV
        lower_bound = np.array([100, 100, 100])  # Example lower bound in HSV
        upper_bound = np.array([140, 255, 255])  # Example upper bound in HSV

        # Create a mask for color segmentation in the HSV space
        mask = cv2.inRange(hsv_frame, lower_bound, upper_bound)

        # Apply the mask to get the segmented image
        segmented_color = cv2.bitwise_and(hsv_frame, hsv_frame, mask=mask)

        # Save the color-segmented image
        color_filename = os.path.join(color_output_folder, f"color_{frame_file}")
        cv2.imwrite(color_filename, segmented_color)

        # Calculate statistics for console output
        num_edges = np.count_nonzero(edges)
        num_segmented_pixels = np.count_nonzero(mask)
        total_pixels = hsv_frame.size // 3  # Each pixel has 3 channels (HSV)

        # Print statistics and display frames for the first few
        if idx < 4:  # Change to the desired number of frames
            print(f"Frame: {frame_file}")
            print(f"  - Total Pixels: {total_pixels}")
            print(f"  - Edge Pixels: {num_edges}")
            print(f"  - Color-Segmented Pixels: {num_segmented_pixels}")

            # Display the original and segmented images
            plt.figure(figsize=(10, 5))

            # Original HSV Frame
            plt.subplot(1, 3, 1)
            plt.imshow(cv2.cvtColor(hsv_frame, cv2.COLOR_HSV2RGB))  # Convert to RGB for display
```

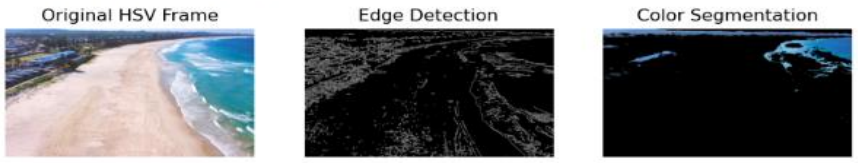
```
Frame: hsv_frame_0000.png
  - Total Pixels: 518400
  - Edge Pixels: 40584
  - Color-Segmented Pixels: 42661
```




```
----------------------------------------
Frame: hsv_frame_0001.png
  - Total Pixels: 518400
  - Edge Pixels: 39116
  - Color-Segmented Pixels: 42971
```



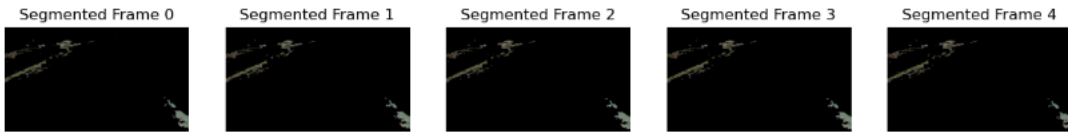

```
----------------------------------------
Frame: hsv_frame_0002.png
  - Total Pixels: 518400
  - Edge Pixels: 39479
  - Color-Segmented Pixels: 42664
```




```
----------------------------------------
```

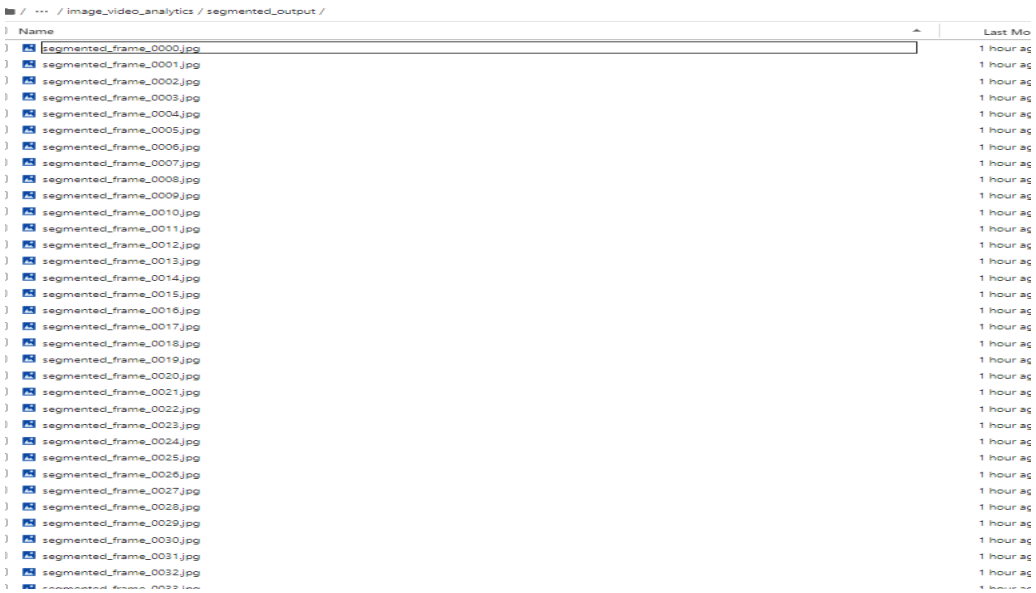
- Track the segmented objects across frames to observe changes in motion and shape.

```
# Define color range for segmentation in HSV (modify according to your need)
lower_bound = np.array([10, 20, 30])
upper_bound = np.array([80, 155, 155])

segmented_folder = 'segmented_frames'
if not os.path.exists(segmented_folder):
    os.makedirs(segmented_folder)

segmented_frames = []

for i, hsv_frame in enumerate(hsv_frames):
    # Threshold the HSV image to get only selected colors
    mask = cv2.inRange(hsv_frame, lower_bound, upper_bound)
    segmented_frame = cv2.bitwise_and(frames_output[i], frames_output[i], mask=mask)
    segmented_filename = os.path.join(segmented_folder, f'segmented_frame_{i:04d}.png')
    cv2.imwrite(segmented_filename, segmented_frame)
    segmented_frames.append(segmented_frame)

# Plot the first 5 segmented frames
plt.figure(figsize=(15, 5))
for i in range(5):
    plt.subplot(1, 5, i + 1)
    plt.imshow(cv2.cvtColor(segmented_frames[i], cv2.COLOR_BGR2RGB))
    plt.axis('off')
    plt.title(f'Segmented Frame {i}')
plt.show()
```



- Identify the regions that remain consistent over time (foreground vs. background segmentation).

```python
# Function to perform foreground-background segmentation
def segment_foreground_background(input_folder, output_folder):
    # Create output directory for segmented frames and background
    if not os.path.exists(output_folder):
        os.makedirs(output_folder)

    # Get a list of frame files
    frame_files = sorted([f for f in os.listdir(input_folder) if f.endswith('.jpg')])

    # Create background subtractor
    back_sub = cv2.createBackgroundSubtractorMOG2()

    # Iterate through all extracted frames
    for idx, frame_file in enumerate(frame_files):
        frame_path = os.path.join(input_folder, frame_file)

        # Read the frame image
        frame = cv2.imread(frame_path)

        # Apply background subtraction
        fg_mask = back_sub.apply(frame)

        # Optional: Perform morphological operations to clean up the mask
        kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (5, 5))
        fg_mask = cv2.morphologyEx(fg_mask, cv2.MORPH_OPEN, kernel)
        fg_mask = cv2.morphologyEx(fg_mask, cv2.MORPH_CLOSE, kernel)

        # Save the segmented frame to output folder
        segmented_filename = os.path.join(output_folder, f"segmented_{frame_file}")
        cv2.imwrite(segmented_filename, fg_mask)

        # Get the background image from the background subtractor
        background = back_sub.getBackgroundImage()
        if background is None:  # If the background image is not ready
            background = np.zeros_like(frame)  # Create a blank background if not availabl

        # Display pixel values of the frame, foreground mask, and background image for the
        if idx < 5:
            print(f"Segmented frame saved: {segmented_filename}")
            print("Frame Pixel Values:")
            print(frame)
            print("Foreground Mask Pixel Values:")
            print(fg_mask)
            print("Background Pixel Values:")
            print(background)
            print("\n" + "-"*50 + "\n")

        # Wait for 100 ms to move to the next frame
        if cv2.waitKey(100) & 0xFF == ord('q'):
            break

    cv2.destroyAllWindows()
    print("Foreground and background segmentation complete!")

# Main Code to Run
if __name__ == "__main__":
    input_folder = 'frames_output'  # Folder containing color-segmented frames
    output_folder = 'segmented_output'  # Folder to save segmented frames
```



4. **Scene Cut Detection**:

- Use pixel-based comparison or histogram differences between consecutive frames to detect abrupt changes (hard cuts).

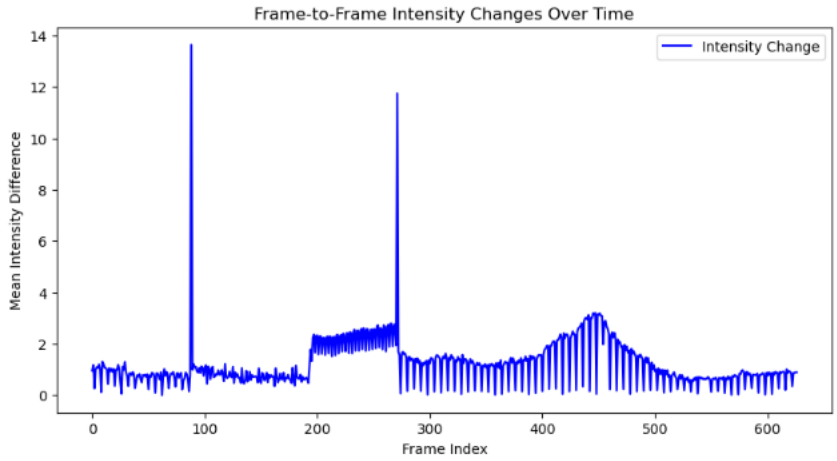
```python
import os
import cv2

# Create directories for histograms, hard cuts, and soft cuts if they don't exist
hist_folder = 'histograms'
hard_cut_folder = 'hard_cuts'
soft_cut_folder = 'soft_cuts'

for folder in [hist_folder, hard_cut_folder, soft_cut_folder]:
    if not os.path.exists(folder):
        os.makedirs(folder)

similarity_scores = []
threshold_hard_cut = 0.5  # Adjust this threshold for hard cuts
soft_cut_threshold = 0.8    # Adjust this for soft cuts

hard_cuts = []
soft_cuts = []

# Function to calculate histogram similarity
def calculate_histogram_similarity(frame1, frame2):
    # Calculate histograms
    hist1 = cv2.calcHist([frame1], [0], None, [256], [0, 256])
    hist2 = cv2.calcHist([frame2], [0], None, [256], [0, 256])

    # Normalize histograms
    hist1 = cv2.normalize(hist1, hist1).flatten()
    hist2 = cv2.normalize(hist2, hist2).flatten()

    # Calculate correlation similarity (0 to 1)
    score = cv2.compareHist(hist1, hist2, cv2.HISTCMP_CORREL)
    return score

# Compare consecutive frames for cuts
for i in range(1, len(frames_output)):
    similarity = calculate_histogram_similarity(frames_output[i-1], frames_output[i])
    similarity_scores.append(similarity)

    # Save histograms
    hist_filename = os.path.join(hist_folder, f'hist_{i:04d}.png')
    hist_img = cv2.calcHist([frames_output[i]], [0], None, [256], [0, 256])
    hist_img = cv2.normalize(hist_img, hist_img).astype('uint8')
    cv2.imwrite(hist_filename, hist_img)

    # Hard Cut Detection
    if similarity < threshold_hard_cut:
        hard_cuts.append(i)
        hard_cut_filename = os.path.join(hard_cut_folder, f'hard_cut_{i:04d}.png')
        cv2.imwrite(hard_cut_filename, frames_output[i])

    # Soft Cut Detection
    elif threshold_hard_cut <= similarity <= soft_cut_threshold:
        soft_cuts.append(i)
        soft_cut_filename = os.path.join(soft_cut_folder, f'soft_cut_{i:04d}.png')
        cv2.imwrite(soft_cut_filename, frames_output[i])

# Display similarity scores for all frames
for i in range(len(similarity_scores)):
```

```python
    # Hard Cut Detection
    if similarity < threshold_hard_cut:
        hard_cuts.append(i)
        hard_cut_filename = os.path.join(hard_cut_folder, f'hard_cut_{i:04d}.png')
        cv2.imwrite(hard_cut_filename, frames_output[i])

    # Soft Cut Detection
    elif threshold_hard_cut <= similarity <= soft_cut_threshold:
        soft_cuts.append(i)
        soft_cut_filename = os.path.join(soft_cut_folder, f'soft_cut_{i:04d}.png')
        cv2.imwrite(soft_cut_filename, frames_output[i])

# Display similarity scores for all frames
for i in range(len(similarity_scores)):
    print(f"Frame {i+1} vs Frame {i+2} Similarity Score: {similarity_scores[i]:.4f}")

# Print total counts of cuts
print(f"Total Hard Cuts Detected: {len(hard_cuts)}")
print(f"Total Soft Cuts Detected: {len(soft_cuts)}")
```

```
Frame 1 vs Frame 2 Similarity Score: 0.9996
Frame 2 vs Frame 3 Similarity Score: 0.9997
Frame 3 vs Frame 4 Similarity Score: 1.0000
Frame 4 vs Frame 5 Similarity Score: 0.9997
Frame 5 vs Frame 6 Similarity Score: 0.9994
Frame 6 vs Frame 7 Similarity Score: 0.9995
Frame 7 vs Frame 8 Similarity Score: 0.9996
Frame 8 vs Frame 9 Similarity Score: 0.9995
Frame 9 vs Frame 10 Similarity Score: 0.9999
Frame 10 vs Frame 11 Similarity Score: 0.9994
Frame 11 vs Frame 12 Similarity Score: 0.9996
Frame 12 vs Frame 13 Similarity Score: 0.9995
Frame 13 vs Frame 14 Similarity Score: 0.9995
Frame 14 vs Frame 15 Similarity Score: 0.9996
Frame 15 vs Frame 16 Similarity Score: 0.9998
Frame 16 vs Frame 17 Similarity Score: 0.9993
Frame 17 vs Frame 18 Similarity Score: 0.9995
Frame 18 vs Frame 19 Similarity Score: 0.9995
```

- Detect gradual scene transitions (Soft cuts) by analyzing frame-to-frame intensity changes over time.

```python
# Main Code to Run
if __name__ == "__main__":
    input_folder = 'color_output'  # Folder containing color-segmented frames
    window_size = 5  # Number of frames to average over for detecting soft cuts
    threshold = 15   # Threshold for detecting soft cuts (tune this value as needed)

    # Detect soft cuts based on frame-to-frame intensity changes
    soft_cuts = detect_soft_cuts(input_folder, window_size, threshold)

    # Print all detected soft cuts
    print("Detected Soft Cuts:")
    for idx, frame_name in soft_cuts:
        print(f"Frame {idx}: {frame_name}")
```

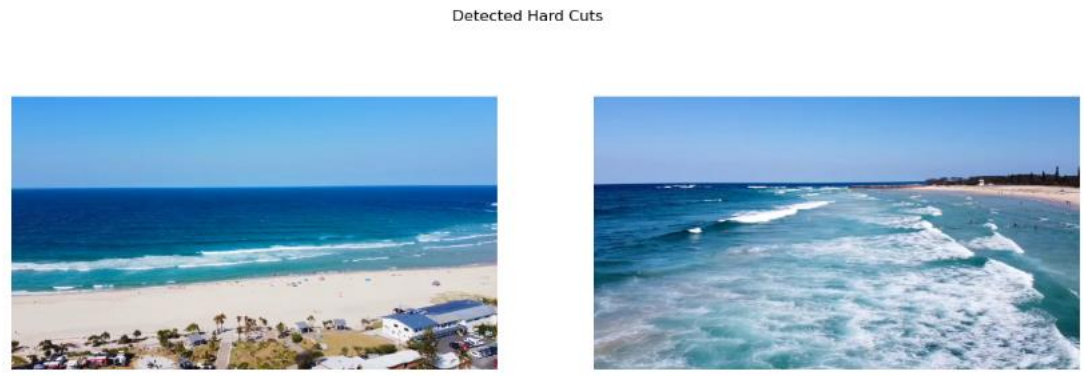


5. Mark Scene Cuts:

a. Highlight the frames where scene cuts are detected.

b. Create a summary displaying the detected scene boundaries.

```python
# Result Visualization
# Function to display frames
def display_frames(frames, title):
    plt.figure(figsize=(15, 5))
    for i, frame in enumerate(frames):
        plt.subplot(1, len(frames), i + 1)
        plt.imshow(cv2.cvtColor(frame, cv2.COLOR_BGR2RGB))
        plt.axis('off')
    plt.suptitle(title)
    plt.show()

# Display detected hard and soft cuts
hard_cut_frames = [frames_output[i] for i in hard_cuts]
soft_cut_frames = [frames_output[i] for i in soft_cuts]

display_frames(hard_cut_frames, "Detected Hard Cuts")
display_frames(soft_cut_frames, "Detected Soft Cuts")
```

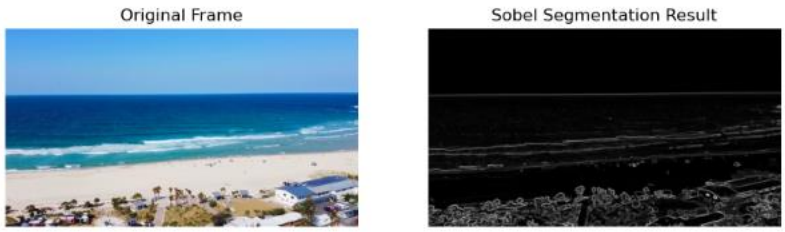


<Figure size 1500x500 with 0 Axes>

6. Result Visualization:

- Display frames where scene cuts are identified and show segmentation results for selected frames.

```python
# Optional: Apply Sobel segmentation and display results for selected frames (e.g., first hard cut)
def sobel_segmentation(frame):
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    sobel_x = cv2.Sobel(gray, cv2.CV_64F, 1, 0, ksize=5)
    sobel_y = cv2.Sobel(gray, cv2.CV_64F, 0, 1, ksize=5)
    sobel_magnitude = cv2.magnitude(sobel_x, sobel_y)
    return sobel_magnitude

# Visualize segmentation results for the first hard cut
if hard_cuts:
    selected_frame_index = hard_cuts[0]
    segmented_frame = sobel_segmentation(frames_output[selected_frame_index])
    plt.figure(figsize=(10, 5))
    plt.subplot(1, 2, 1)
    plt.imshow(cv2.cvtColor(frames_output[selected_frame_index], cv2.COLOR_BGR2RGB))
    plt.title("Original Frame")
    plt.axis('off')
    plt.subplot(1, 2, 2)
    plt.imshow(segmented_frame, cmap='gray')
    plt.title("Sobel Segmentation Result")
    plt.axis('off')
    plt.show()
```



RESULT:

The analysis of the provided video involved several systematic steps, yielding insights into scene cuts and object segmentation:

1. **Frame Extraction**:
    - A total of **627 frames** were extracted and stored for comprehensive analysis of the video content.
2. **Spatio-Temporal Segmentation**:
    - We applied segmentation techniques, including **color thresholding** and **edge detection**, to differentiate between foreground and background elements. This allowed us to track changes in motion and shape over time.
3. **Scene Cut Detection**:
    - **Hard Cut Detection**: Utilizing histogram intersections between consecutive frames, we identified **2 hard cuts**, indicating abrupt scene transitions.
    - **Soft Cut Detection**: No soft cuts were detected, suggesting that the video primarily features abrupt changes rather than gradual transitions.
4. **Marking Scene Cuts**:
    - Detected scene cuts were highlighted, and a summary of scene boundaries was created, facilitating visual analysis of the video's narrative flow.

5. **Result Visualization**:
    - The original frames, along with their edge-detected and color-segmented versions, were displayed to provide clear visual context for the identified cuts.

DISCUSSION

The analysis of the video involved extracting **627 frames** and employing segmentation techniques, notably color thresholding and edge detection, to differentiate between foreground objects and the background. This allowed for a deeper understanding of the video's dynamics.

The detection of **2 hard cuts** indicates significant narrative shifts, while the absence of **soft cuts** suggests a more straightforward pacing. Marking these scene cuts facilitates visual analysis, helping to enhance viewer engagement by clarifying pacing and structure.

Visualizing original frames alongside their segmented versions demonstrated the effectiveness of the segmentation techniques, highlighting their role in comprehending complex narratives. Overall, the methods applied provided a comprehensive approach to video analysis, revealing both technical and artistic insights.

CONCLUSION

Summary of Work: In this assignment, we analyzed a video by extracting 627 frames and applying various segmentation techniques, including color thresholding and edge detection. We successfully detected 2 hard cuts and identified no soft cuts, enabling us to mark significant transitions in the video. Additionally, we visualized the original and processed frames to enhance our understanding of the content and structure.

Key Takeaways: This project highlighted the importance of effective video analysis techniques in understanding narrative flow and scene transitions. The segmentation methods allowed us to differentiate between dynamic foreground elements and static backgrounds. Furthermore, detecting scene cuts provided insight into pacing, emphasizing the significance of both hard and soft cuts in video editing and storytelling.

GIT HUB LINK: https://github.com/Vishali2002/Spatio--Temporal-segmentation.git