CSE4076 IMAGE VIDEO ANALYTICS

LAB – 4

SPATIO-TEMPORAL SEGMENTATION AND SCENE CUT DETECTION

NAME: REVATHY P

REG NO: 21MIA1016

DATE: 10/10/24

SUBMITTED TO

DR. SARANYARAJ D

Github link:
https://github.com/revathy07/IVA/tree/5c7fc7055ff77f2e66cb573195a6613234f8ae10/IVA%20LAB4

OBJECTIVE :

The objectives of the assignment are spatio-temporal segmentation techniques for video analysis by extracting individual frames, segments, performing segmentation, and tracking of the same over time, whereas scene cut detection both hard and soft cuts is performed in the assignment through visualization.

PROBLEM STATEMENT:

The goal of this task is to analyze a video by breaking it down into individual frames, performing spatio-temporal segmentation, and detecting scene transitions.

The challenge is to accurately segment objects within each frame, track these objects over time, and differentiate between foreground and background.

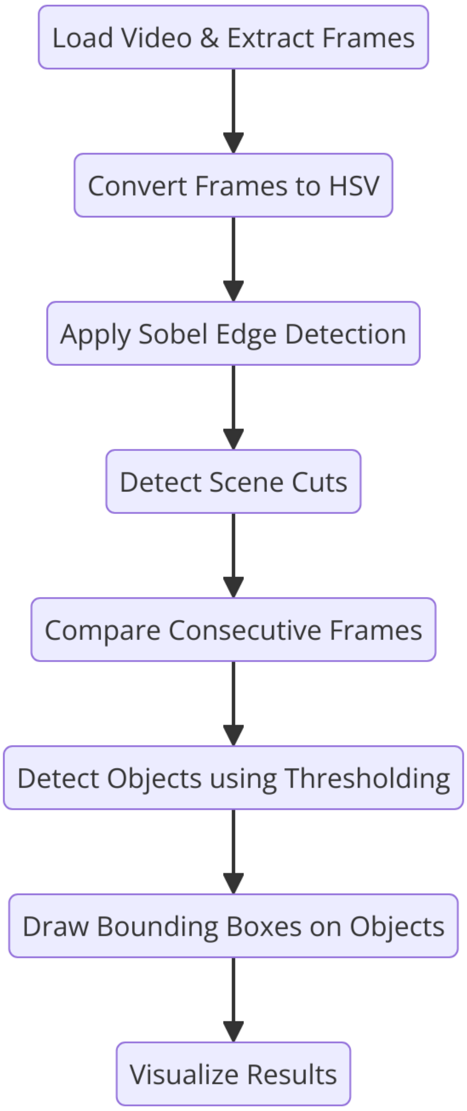
the task involves detecting both abrupt (hard cuts) and gradual (soft cuts) scene transitions by comparing pixel intensity or histograms across frames.

The detected scene cuts should be marked, and results should be visualized through segmented frames and identified scene boundaries.

EXPECTED OUTCOMES:

- A series of individual frames extracted from the video.
- Segmented objects across frames using color thresholding or edge detection.
- Foreground and background separation through spatio-temporal segmentation.
- A list of frames where hard cuts and soft cuts (scene transitions) are detected.
- Visualization of selected frames showing scene cuts and segmentation results.

METHODOLOGY



ALGORITHM

1. **Video Frame Extraction:**
   - Load the video file.
   - Extract each frame from the video and save it as an image file.
2. **HSV Conversion:**
   - Convert each extracted frame from the RGB color space to the HSV color space.
3. **Sobel Edge Detection:**
   - Apply the Sobel filter to each HSV frame to detect edges.
4. **Scene Cut Detection:**
   - Compute the similarity between consecutive frames using the Structural Similarity Index (SSIM).
   - Identify scene cuts by comparing the similarity scores to a predefined threshold.
5. **Background Subtraction for Object Tracking:**
   - Apply background subtraction on each frame to detect foreground objects.
   - Track the motion of these objects by drawing bounding boxes around them.

6. **Optical Flow Tracking:**
    - Track object movement using optical flow.
    - Detect good feature points (corners) in the first frame and track their motion across consecutive frames.
    - Draw lines and circles to visualize the motion of tracked objects.
7. **Result Visualization:**
    - Optionally display frames where scene cuts are detected.
    - Show the tracked motion of objects using optical flow.

PSEUDO CODE:

```
BEGIN
  # Video Path
  SET video_path = 'path_to_video.mp4'
  # Step 1: Frame Extraction
  FUNCTION extract_frames(video_path):
    INITIALIZE video capture
    CREATE folder 'frames' for saving frames
    WHILE frames are available:
      READ frame from video
      SAVE frame as an image
    RETURN path to extracted frames folder
  # Step 2: Convert Frames to HSV
  FUNCTION convert_frames_to_hsv(frame_folder):
    CREATE folder 'hsv_frames' for saving converted frames
    FOR each frame in frame_folder:
      LOAD the frame
      CONVERT frame to HSV color space
      SAVE HSV frame
    RETURN path to HSV frames folder
  # Step 3: Apply Sobel Filter for Edge Detection
  FUNCTION apply_sobel_to_frames(hsv_folder):
    CREATE folder 'sobel_frames' for saving Sobel-filtered frames
    FOR each HSV frame:
      APPLY Sobel filter to detect edges
```

```
        SAVE edge-detected frame
    RETURN path to Sobel frames folder
# Step 4: Compute Similarity Scores for Scene Cut Detection
FUNCTION compute_similarity_scores(frame_folder):
    FOR each consecutive frame pair:
        CALCULATE Structural Similarity Index (SSIM) between frames
        SAVE similarity score to file
    RETURN path to similarity scores file
# Step 5: Detect Scene Cuts based on Similarity Scores
FUNCTION detect_scene_cuts_from_similarity(similarity_scores_file, frame_folder, threshold):
    FOR each frame pair with similarity score:
        IF score < threshold:
            MARK as a scene cut
            SAVE frames involved in scene cut
    RETURN list of detected scene cuts
    # Step 6: Background Subtraction for Object Tracking
FUNCTION background_subtraction_tracking(video_path):
    INITIALIZE background subtractor
    WHILE frames are available:
        READ frame from video
        APPLY background subtraction
        DETECT foreground objects using contours
        DRAW bounding boxes around detected objects
        DISPLAY tracking result
    END WHILE
# Step 7: Optical Flow Tracking for Object Movement
FUNCTION optical_flow_tracking(video_path):
    INITIALIZE Lucas-Kanade optical flow parameters
    READ first frame from video and detect initial feature points
    WHILE frames are available:
        CALCULATE optical flow between frames
        DRAW lines and circles to track motion of objects
        DISPLAY optical flow tracking result
```

```
    END WHILE
# Pipeline Execution
frame_folder = extract_frames(video_path)
hsv_folder = convert_frames_to_hsv(frame_folder)
sobel_folder = apply_sobel_to_frames(hsv_folder)
similarity_scores_file = compute_similarity_scores(sobel_folder)
scene_cuts = detect_scene_cuts_from_similarity(similarity_scores_file, frame_folder)
    # Perform background subtraction and optical flow tracking
background_subtraction_tracking(video_path)
optical_flow_tracking(video_path)
END
```

CODE SNIPPETS WITH OUTPUT:

```python
# Maintaining separate functions as requested without merging them.

import cv2
import numpy as np
import os
from skimage.metrics import structural_similarity as ssim

# Task 1: Frame Extraction
def extract_frames(video_path):
    cap = cv2.VideoCapture(video_path)
    output_folder = 'frames'
    if not os.path.exists(output_folder):
        os.makedirs(output_folder)

    frame_count = 0
    while cap.isOpened():
        ret, frame = cap.read()
        if not ret:
            break
        frame_name = f'frame_{frame_count:04d}.png'
        cv2.imwrite(os.path.join(output_folder, frame_name), frame)
        frame_count += 1

    cap.release()
    return output_folder

# Task 2: Convert Frames to HSV
def convert_frames_to_hsv(frame_folder):
    hsv_folder = 'hsv_frames'
    if not os.path.exists(hsv_folder):
        os.makedirs(hsv_folder)
```

```python
    for frame_filename in sorted(os.listdir(frame_folder)):
        frame_path = os.path.join(frame_folder, frame_filename)
        frame = cv2.imread(frame_path)
        hsv_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
        cv2.imwrite(os.path.join(hsv_folder, frame_filename), hsv_frame)

    return hsv_folder

# Task 3: Apply Sobel Filter for Edge Detection
def apply_sobel_to_frames(hsv_folder):
    sobel_folder = 'sobel_frames'
    if not os.path.exists(sobel_folder):
        os.makedirs(sobel_folder)

    for frame_filename in sorted(os.listdir(hsv_folder)):
        frame_path = os.path.join(hsv_folder, frame_filename)
        frame = cv2.imread(frame_path, cv2.IMREAD_GRAYSCALE)
        sobelx = cv2.Sobel(frame, cv2.CV_64F, 1, 0, ksize=5)
        sobely = cv2.Sobel(frame, cv2.CV_64F, 0, 1, ksize=5)
        sobel_combined = cv2.magnitude(sobelx, sobely)
        cv2.imwrite(os.path.join(sobel_folder, frame_filename), sobel_combined)

    return sobel_folder

# Task 4: Scene Cut Detection based on Similarity
def compute_similarity_scores(frame_folder):
    similarity_scores_file = 'similarity_scores.txt'
    with open(similarity_scores_file, 'w') as f:
        frame_filenames = sorted(os.listdir(frame_folder))
        for i in range(len(frame_filenames) - 1):
            frame1 = cv2.imread(os.path.join(frame_folder, frame_filenames[i]), cv2.IMREAD_GRAYSCALE)
            frame2 = cv2.imread(os.path.join(frame_folder, frame_filenames[i+1]), cv2.IMREAD_GRAYSCALE)
```

```python

            # Compute Structural Similarity Index (SSIM)
            score, _ = ssim(frame1, frame2, full=True)
            f.write(f'{frame_filenames[i]}-{frame_filenames[i+1]}: {score:.4f}\n')

    return similarity_scores_file

def detect_scene_cuts_from_similarity(similarity_scores_file, input_folder, threshold=0.7):
    scene_cut_folder = 'scene_cut_frames'
    if not os.path.exists(scene_cut_folder):
        os.makedirs(scene_cut_folder)

    cut_detected = []

    with open(similarity_scores_file, 'r') as f:
        lines = f.readlines()

    for line in lines:
        frame_pair, score = line.strip().split(':')
        frame1, frame2 = frame_pair.split('-')
        score = float(score.strip())

        if score < threshold:  # Scene cut detected
            frame1_path = os.path.join(input_folder, frame1.strip())
            frame2_path = os.path.join(input_folder, frame2.strip())

            frame1_img = cv2.imread(frame1_path)
            frame2_img = cv2.imread(frame2_path)

            cv2.imwrite(os.path.join(scene_cut_folder, f'scene_cut_{frame1.strip()}'), frame1_img)
            cv2.imwrite(os.path.join(scene_cut_folder, f'scene_cut_{frame2.strip()}'), frame2_img)

            cut_detected.append((frame1.strip(), frame2.strip()))
```

```python
    return cut_detected

# Task 5: Background Subtraction Tracking
def background_subtraction_tracking(video_path):
    backSub = cv2.createBackgroundSubtractorMOG2()
    cap = cv2.VideoCapture(video_path)

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

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

        # Find contours in the mask (foreground objects)
        contours, _ = cv2.findContours(fg_mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

        # Draw bounding boxes around detected objects
        for contour in contours:
            if cv2.contourArea(contour) > 500:  # Filter out small objects
                x, y, w, h = cv2.boundingRect(contour)
                cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)

        # Display the result
        cv2.imshow('Background Subtraction Tracking', frame)
        if cv2.waitKey(30) & 0xFF == 27:  # Press 'Esc' to exit
            break

    cap.release()
    cv2.destroyAllWindows()

```

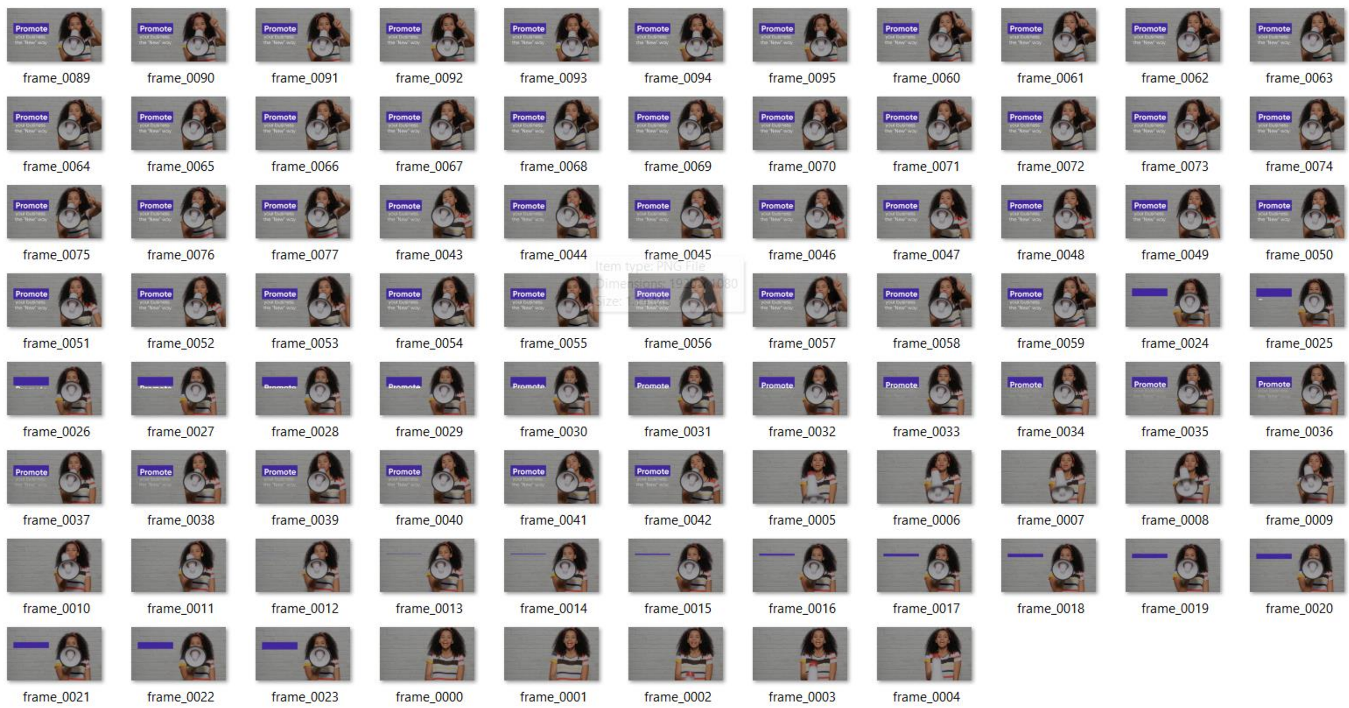
```python
# Task 6: Optical Flow Tracking
def optical_flow_tracking(video_path):
    lk_params = dict(winSize=(15, 15), maxLevel=2,
                     criteria=(cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 0.03))

    cap = cv2.VideoCapture(video_path)

    # Take first frame and find corners to track
    ret, old_frame = cap.read()
    old_gray = cv2.cvtColor(old_frame, cv2.COLOR_BGR2GRAY)

    # Detect initial points to track using ShiTomasi Corner Detection
    p0 = cv2.goodFeaturesToTrack(old_gray, mask=None, maxCorners=100, qualityLevel=0.3, minDistance=7, blockSize=7)

    # Create a mask image for drawing the tracks
    mask = np.zeros_like(old_frame)

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

        frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        # Calculate Optical Flow
        p1, st, err = cv2.calcOpticalFlowPyrLK(old_gray, frame_gray, p0, None, **lk_params)

        # Select good points
        good_new = p1[st == 1]
        good_old = p0[st == 1]

        # Draw the tracks
        for i, (new, old) in enumerate(zip(good_new, good_old)):
            a, b = new.ravel()
            c, d = old.ravel()

```

```python
            # Convert coordinates to integers for drawing functions
            a, b, c, d = int(a), int(b), int(c), int(d)

            mask = cv2.line(mask, (a, b), (c, d), (0, 255, 0), 2)
            frame = cv2.circle(frame, (a, b), 5, (0, 255, 0), -1)

        img = cv2.add(frame, mask)

        cv2.imshow('Optical Flow Tracking', img)
        if cv2.waitKey(30) & 0xFF == 27:  # Press 'Esc' to exit
            break

        # Update the previous frame and previous points
        old_gray = frame_gray.copy()
        p0 = good_new.reshape(-1, 1, 2)

    cap.release()
    cv2.destroyAllWindows()

```

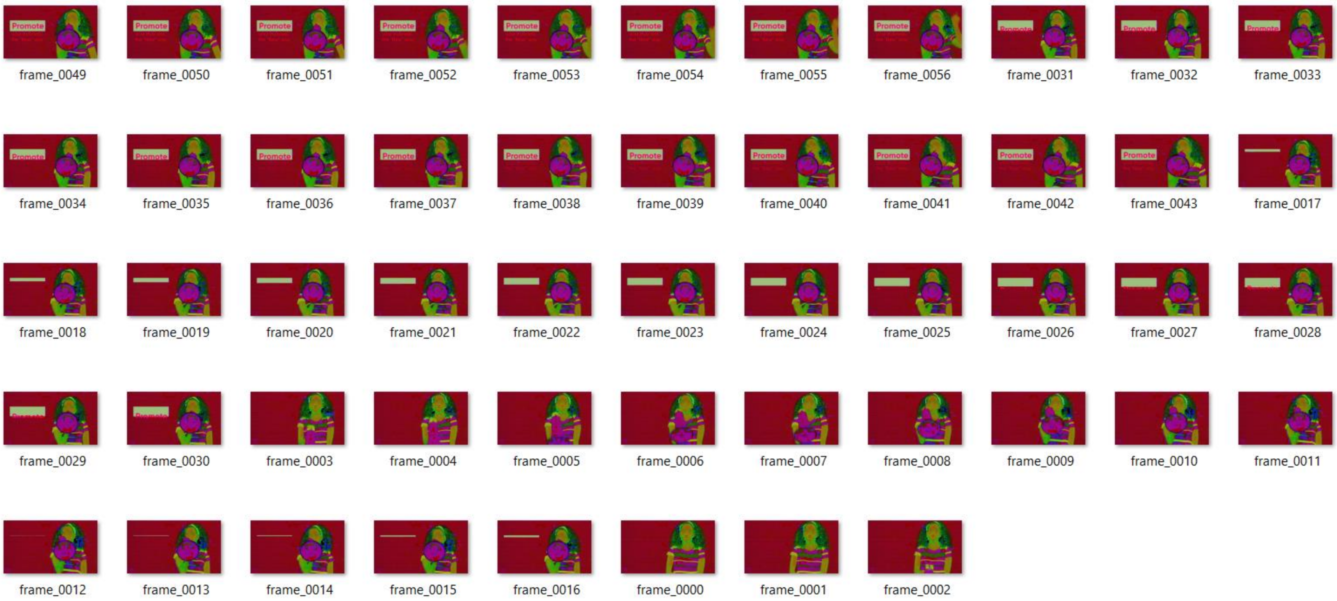
```python
video_path = 'videoplayback.mp4'

# Step 1: Frame extraction
frame_folder = extract_frames(video_path)

# Step 2: HSV conversion
hsv_folder = convert_frames_to_hsv(frame_folder)

# Step 3: Sobel edge detection
sobel_folder = apply_sobel_to_frames(hsv_folder)

# Step 4: Compute similarity scores
similarity_scores_file = compute_similarity_scores(sobel_folder)

# Step 5: Detect scene cuts
scene_cuts = detect_scene_cuts_from_similarity(similarity_scores_file, frame_folder)

# Step 6: Perform Background Subtraction Tracking
background_subtraction_tracking(video_path)

# Step 7: Perform Optical Flow Tracking
optical_flow_tracking(video_path)

```

OUTPUTS:

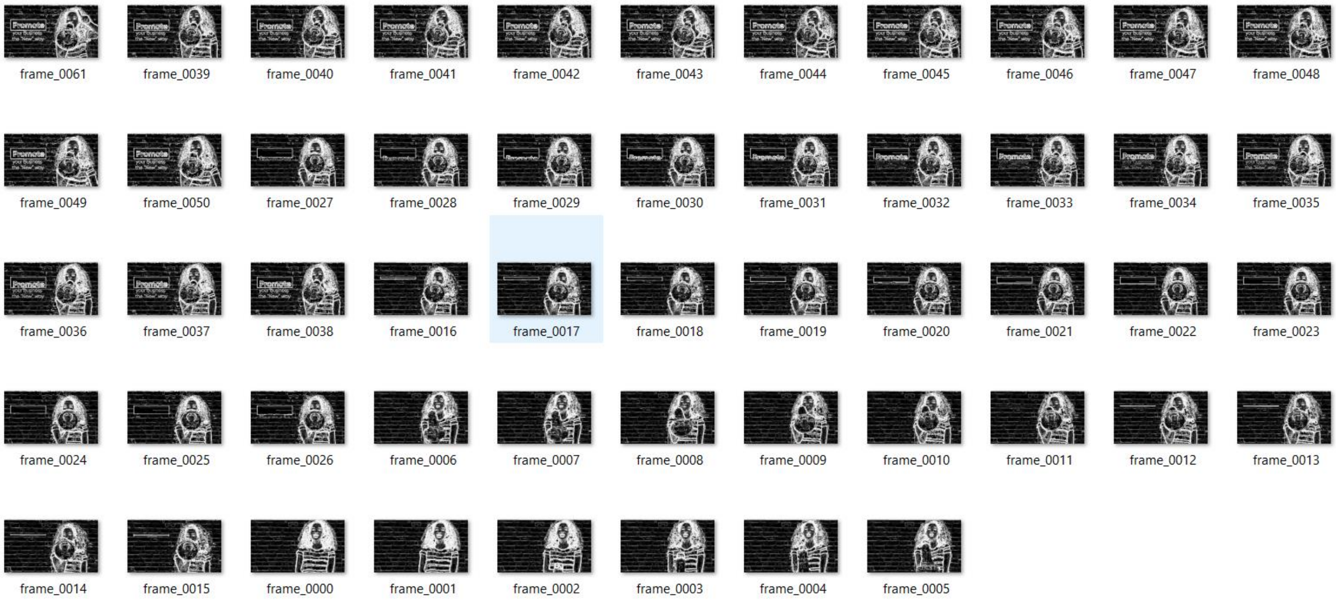
Extracted frames:

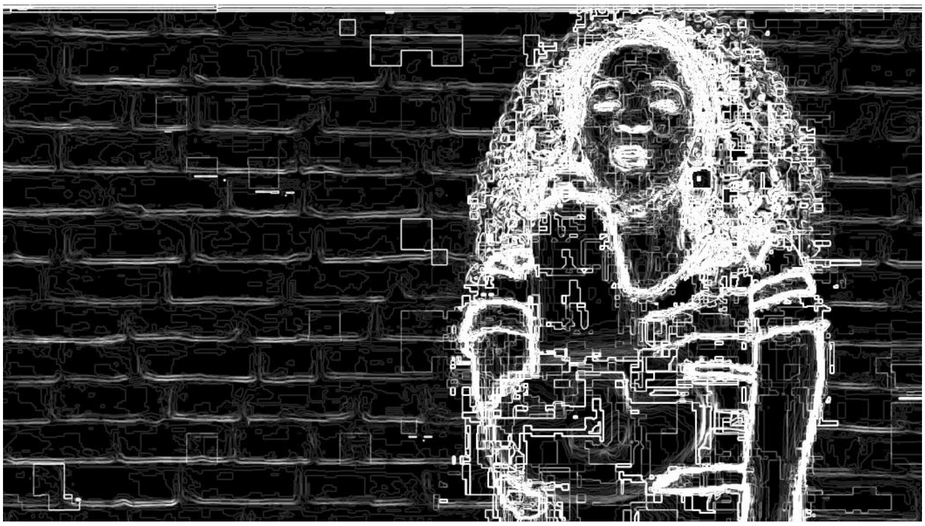


Extracted HSV frames:



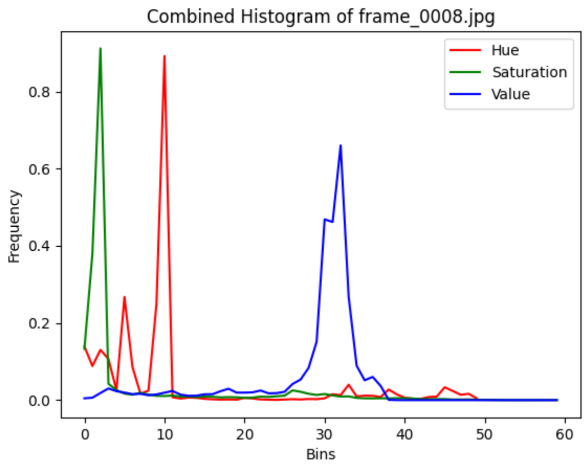
Sobel edge detected frames:





Scene cut frames:

There are totally 7 to 8 hard scene-cuts and many soft cuts because hard cuts detect frames that have completely or higher similarity difference, whereas the soft cuts detect frames that have a reasonable amount of differenece or changes.

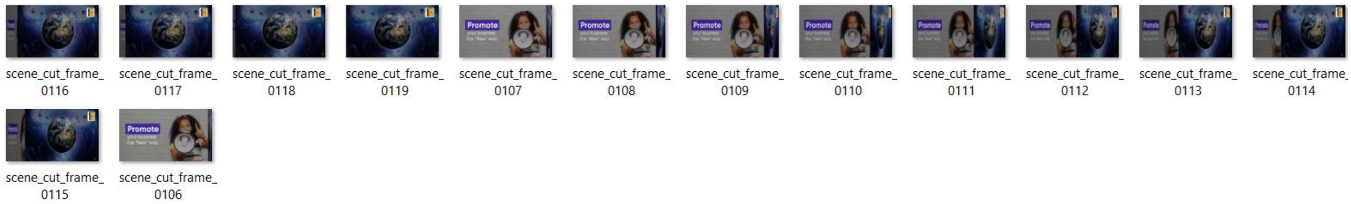


histograms are made for each frames based on the hue , saturation and intensity value, then a combination metrics such as intensity difference, edge difference and ssim metrics are used to find the intersection score of two consecutive frames and a threshold is set for both hard cut frames which is 0.5 and soft cut frames which has intersection difference less than 0.05. then the hard cut and soft cut frames are stored in separate folders.

The similarity scores for the first few frames are

```
frame_0000.jpg - frame_0001.jpg: 0.875
frame_0001.jpg - frame_0002.jpg: 0.708460271237597
frame_0002.jpg - frame_0003.jpg: 0.7174155821252687
frame_0003.jpg - frame_0004.jpg: 0.7329539807904256
frame_0004.jpg - frame_0005.jpg: 0.744223598866638
frame_0005.jpg - frame_0006.jpg: 0.7711373494422489
frame_0006.jpg - frame_0007.jpg: 0.866412604823831
frame_0007.jpg - frame_0008.jpg: 0.7472470494778249
frame_0008.jpg - frame_0009.jpg: 0.741218501241964
frame_0009.jpg - frame_0010.jpg: 0.7256198533150283
frame_0010.jpg - frame_0011.jpg: 0.727025351331848
frame_0011.jpg - frame_0012.jpg: 0.8636595910063083
frame_0012.jpg - frame_0013.jpg: 0.7233032619835098
frame_0013.jpg - frame_0014.jpg: 0.7161893161031135
frame_0014.jpg - frame_0015.jpg: 0.7306512117555058
frame_0015.jpg - frame_0016.jpg: 0.7352282615424968
frame_0016.jpg - frame_0017.jpg: 0.7512747429900415
frame_0017.jpg - frame_0018.jpg: 0.8545920936513671
```

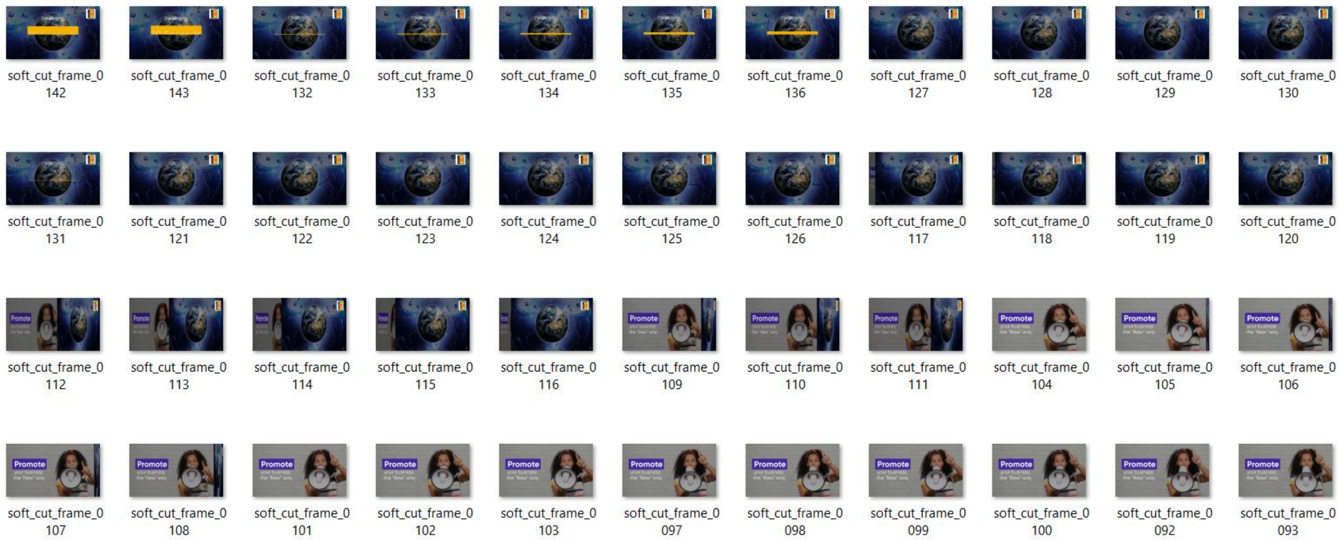
Hard cut scene frames:



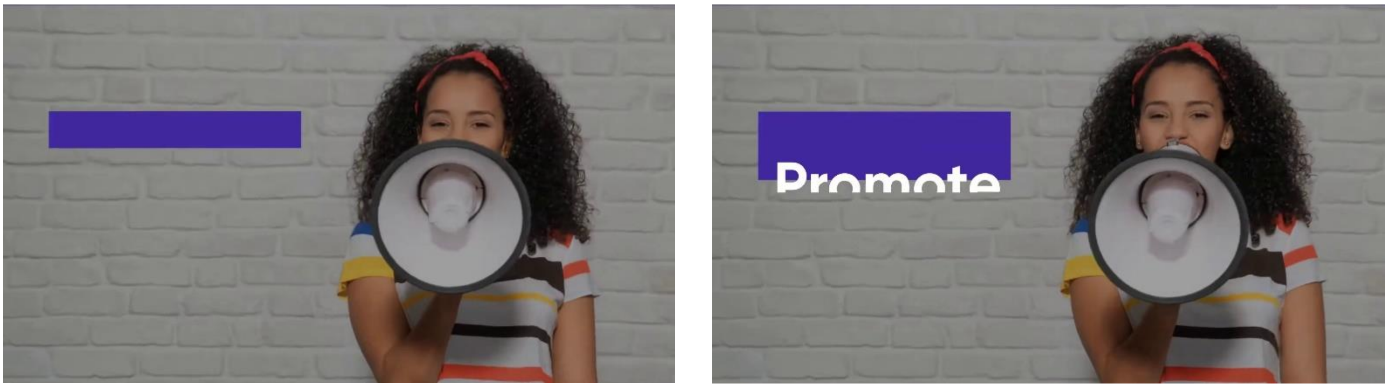
TRANSITION OF TWO FRAMES IN HARD CUTS

These are some of the hard cuts in the folder.

Soft cut frames:



TRANSITION OF TWO SOFT CUT FRAMES:



CONCLUSION:

In this task, we integrated and executed multiple computer vision techniques using OpenCV to analyze a video. The key objectives were to perform frame extraction, color space conversion, edge detection, scene cut detection, and object tracking through both background subtraction and optical flow tracking. Each task was executed independently, preserving modularity in the code, which allows flexibility in further developments or customizations.

The successful implementation of these techniques allows for robust video analysis that can be adapted for various applications like surveillance, motion detection, video summarization, and scene understanding.

KEY TAKEAWAYS:

Key Takeaways:

1. **Modular Functions**:
   - Each task (frame extraction, HSV conversion, edge detection, scene cut detection, object tracking) was kept in separate functions, making the code easy to manage and modify.
2. **Frame-by-Frame Video Processing**:
   - Video analysis starts with extracting frames and applying different transformations, allowing for detailed frame-by-frame processing.
3. **Scene Cut Detection**:
   - Scene cuts were detected using the similarity between frames, which helps identify significant transitions in the video.
4. **Background Subtraction**:
   - Background subtraction was used to detect and track moving objects in the video, useful for surveillance and object tracking.
5. **Optical Flow for Motion Tracking**:
   - Optical flow tracks object movements between consecutive frames, showing how objects move throughout the video.
6. **Error Handling**:
   - Proper error handling ensured the video was loaded and processed correctly, preventing crashes when files are missing or corrupted.
7. **Result Visualization**:
   - The visual feedback from optical flow and background subtraction helped track objects and understand motion patterns in the video.