IMAGE VIDEO ANALYTICS - CSE4076

LAB – 4

SPATIO – TEMPORAL SEGMENTATION

NAME: M.SUSILASHA

REG NO: 21MIA1006

DATE: 10/10/24

SUBMITTED TO

DR. SARANYARAJ D

1. GITHUB LINK :
https://github.com/Susilasha02/IVA_LAB_21MIA1006_M.SUSILASHA/tree/main/IVA_LAB4

2. Objective

The primary goal of this project is to use several image processing algorithms on a video to extract frames, segment it spatiotemporally, and detect scene cuts. The assignment will also contain object tracking between frames using edge detection and color thresholding, as well as scene cuts marked with similarity measures such as the Structural Similarity Index (SSIM), pixel intensity differences, and histogram comparisons.

3. Problem Statement

- **Description of the Problem:**

The challenge entails first processing a video to extract individual frames, and then using segmentation techniques to track and detect objects inside those frames. One of the most difficult issues is distinguishing sudden changes between scenes (hard cuts) and smooth transitions (soft cuts) by comparing the similarities of successive frames. In addition, spatiotemporal segmentation will be used to recognize items that move between frames and distinguish between foreground and background.

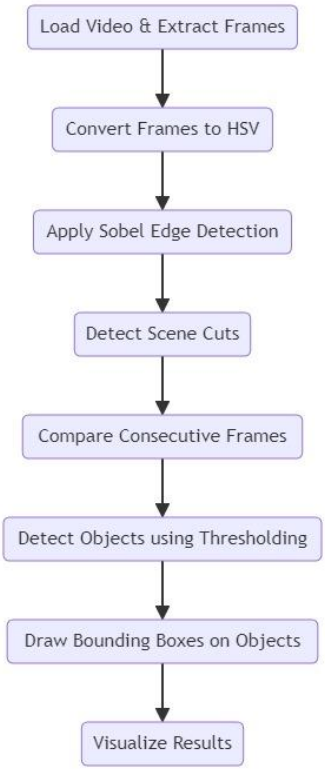
- **Expected Output:**

The final output should include:

1. Extracted frames from the video.
2. Sobel edge-detected frames showing object boundaries.
3. Scene cut detection results with marked frames indicating both hard and soft cuts.
4. Frames with bounding boxes around tracked objects.
5. A visualization of frames where scene cuts were detected.

4. Methodology

- **Block Diagram**



- **Algorithm**

Algorithm: The task follows the below steps:

1. **Load Video and Frame Extraction**:
    - Open the video file using OpenCV and extract individual frames.
    - Save each frame as a separate image file for further processing.
2. **Convert Frames to HSV**:
    - Convert extracted frames from BGR color space to HSV for better segmentation performance when performing thresholding based on color.
3. **Spatio-Temporal Segmentation using Sobel Edge Detection**:
    - Convert frames to grayscale and apply Sobel edge detection to identify object boundaries.
    - Use gradients along the x and y axes to compute edge maps.
4. **Scene Cut Detection using SSIM and Histograms**:
    - Use SSIM to measure the structural similarity between consecutive frames.
    - Compute pixel intensity differences between frames.
    - Compare histograms of frames to detect significant differences in brightness or color distribution.
    - Mark frames where similarity drops below a threshold, indicating potential scene cuts.

5. **Object Tracking**:
    - Apply color thresholding in the HSV color space to detect objects (e.g., red objects).
    - Find contours around detected objects and draw bounding boxes around them to track object motion.
6. **Result Visualization**:
    - Display and save the frames where scene cuts and objects were detected.

- **Pseudo code**

i. Load Video and Extract Frames:

- Use OpenCV's `cv2.VideoCapture()` to load the video.

- Extract individual frames using a loop with the `cap.read()` function.

- Save each frame as an image using `cv2.imwrite()`.

ii. Convert Frames to HSV:

- Convert each extracted frame from BGR to HSV using OpenCV's `cv2.cvtColor()` function.

- Formula: `HSV = cv2.cvtColor(BGR, cv2.COLOR_BGR2HSV)`.

iii. Spatio-Temporal Segmentation using Sobel Edge Detection:

- Convert the frame to grayscale using `cv2.cvtColor()`.

- Apply Sobel edge detection along the x-axis (`cv2.Sobel()`) and y-axis:

- Sobel formula for x-gradient: `Gx = cv2.Sobel(gray_frame, cv2.CV_64F, 1, 0, ksize=3)`.

- Sobel formula for y-gradient: `Gy = cv2.Sobel(gray_frame, cv2.CV_64F, 0, 1, ksize=3)`.

- Combine gradients to get the overall edge magnitude using the formula: `G = sqrt(Gx^2 + Gy^2)`

iv. Scene Cut Detection using SSIM and Histograms:

a) SSIM (Structural Similarity Index):

- SSIM measures the structural similarity between two frames using the formula:

```
```

SSIM(x, y) = (2 * mean_x * mean_y + C1) * (2 * covariance_xy + C2) / ((mean_x^2 + mean_y^2 + C1) * (variance_x + variance_y + C2))

```

- Use the `skimage.metrics.structural_similarity()` function to compute SSIM between two frames.

**b) Pixel Intensity Differences:**

- Compute absolute difference between consecutive frames using OpenCV's `cv2.absdiff()` function.

- Check if the intensity difference exceeds a threshold.

```
intensity_diff = cv2.absdiff(frame1, frame2)
diff_value = np.sum(intensity_diff)
if diff_value > intensity_threshold:
print(f"Scene cut detected at frame {frame_number}")
```

**c) Histogram Comparison:**

- Convert frames to grayscale and compute the histogram using `cv2.calcHist()`.

- Compare the histograms of consecutive frames using correlation or Chi-square distance:

```
histogram_diff = cv2.compareHist(hist1, hist2, cv2.HISTCMP_CORREL)
if histogram_diff < hist_threshold:
    print(f"Scene cut detected at frame {frame_number}")
```

v. Object Detection using Color Thresholding in HSV:

- Define HSV color ranges for objects to be detected, e.g., detecting a red object:

```
lower_bound = np.array([0, 120, 70])
upper_bound = np.array([10, 255, 255])
```

- Use `cv2.inRange()` to apply the color thresholding.

- Find contours of the detected objects using `cv2.findContours()` and draw bounding boxes using `cv2.rectangle()`.

vi. Result Visualization:
```

- Display and save frames with detected objects and scene cuts marked using `cv2.imshow()` and `cv2.imwrite()`.

5. Python Implementation

```
import cv2
import os
import numpy as np
from skimage.metrics import structural_similarity as ssim
import matplotlib.pyplot as plt

# Task 1: Load Video and Frame Extraction
def extract_frames(video_path, output_folder='output_frames'):
    if not os.path.exists(output_folder):
        os.makedirs(output_folder)

    video_capture = cv2.VideoCapture(video_path)

    if not video_capture.isOpened():
        print("Error: Could not open video.")
        return

    frame_count = 0
    while True:
        success, frame = video_capture.read()
        if not success:
            break
        frame_filename = os.path.join(output_folder,
f'frame_{frame_count:04d}.jpg')
        cv2.imwrite(frame_filename, frame)
        frame_count += 1
        print(f"Saved {frame_filename}")

    video_capture.release()
    print(f"All {frame_count} frames have been extracted and saved.")
    return output_folder

# Task 2: Convert Frames to HSV Color Space
def convert_frames_to_hsv(input_folder,
output_hsv_folder='output_hsv_frames'):
    if not os.path.exists(output_hsv_folder):
        os.makedirs(output_hsv_folder)

    for frame_filename in sorted(os.listdir(input_folder)):
        frame_path = os.path.join(input_folder, frame_filename)
        frame = cv2.imread(frame_path)

        # Convert frame to HSV color space
```

```
        hsv_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

        # Save the HSV frame
        hsv_frame_filename = os.path.join(output_hsv_folder, frame_filename)
        cv2.imwrite(hsv_frame_filename, hsv_frame)
        print(f"Saved HSV frame: {hsv_frame_filename}")

    print("All frames have been converted to HSV and saved.")
    return output_hsv_folder

# Task 3: Spatio-Temporal Segmentation (Object Tracking and
Foreground/Background Segmentation)
def background_subtraction_tracking(input_folder,
output_fg_folder='output_fg_tracking'):
    if not os.path.exists(output_fg_folder):
        os.makedirs(output_fg_folder)

    # Use BackgroundSubtractorMOG2 for foreground segmentation
    bg_subtractor = cv2.createBackgroundSubtractorMOG2(history=500,
varThreshold=16, detectShadows=True)

    for frame_filename in sorted(os.listdir(input_folder)):
        frame_path = os.path.join(input_folder, frame_filename)
        frame = cv2.imread(frame_path)

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

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

        # Draw bounding boxes around the 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)

        # Save the foreground tracking frame
        fg_frame_filename = os.path.join(output_fg_folder, frame_filename)
        cv2.imwrite(fg_frame_filename, frame)
        print(f"Saved foreground-tracked frame: {fg_frame_filename}")

    print("Foreground/background segmentation and object tracking completed
for all frames.")
    return output_fg_folder

# Task 4: Scene Cut Detection with Hard and Soft Cut
```

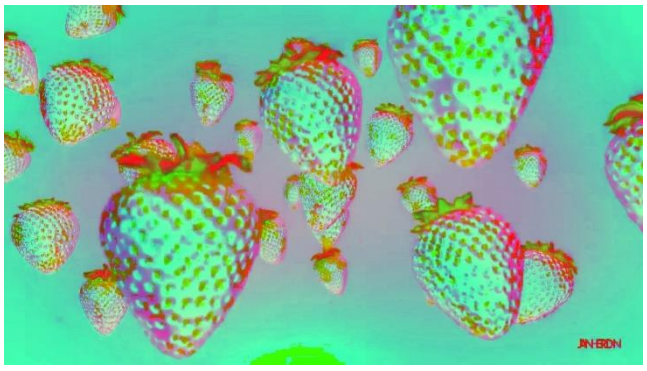
```
def compute_similarity_scores(hsv_frames_folder,
histograms_folder='output_histograms',
similarity_scores_file='similarity_scores.txt'):
    if not os.path.exists(histograms_folder):
        os.makedirs(histograms_folder)

    frame_filenames = sorted(os.listdir(hsv_frames_folder))
    similarity_scores = []

    def sobel_edge_detection(image):
        gray = cv2.cvtColor(image, 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_edges = np.sqrt(sobel_x**2 + sobel_y**2)
        return sobel_edges

    for i in range(len(frame_filenames) - 1):
        frame1_path = os.path.join(hsv_frames_folder, frame_filenames[i])
        frame2_path = os.path.join(hsv_frames_folder, frame_filenames[i + 1])

        frame1 = cv2.imread(frame1_path)
        frame2 = cv2.imread(frame2_path)

        gray_frame1 = cv2.cvtColor(frame1, cv2.COLOR_BGR2GRAY)
        gray_frame2 = cv2.cvtColor(frame2, cv2.COLOR_BGR2GRAY)

        score_ssim, _ = ssim(gray_frame1, gray_frame2, full=True)

        intensity_diff = np.mean(np.abs(gray_frame1.astype("float") -
gray_frame2.astype("float")))

        sobel_frame1 = sobel_edge_detection(frame1)
        sobel_frame2 = sobel_edge_detection(frame2)

        sobel_diff = np.mean(np.abs(sobel_frame1 - sobel_frame2))

        hist1_value = cv2.calcHist([frame1], [2], None, [60], [0, 256])
        hist2_value = cv2.calcHist([frame2], [2], None, [60], [0, 256])

        hist1_value = cv2.normalize(hist1_value, hist1_value).flatten()
        hist2_value = cv2.normalize(hist2_value, hist2_value).flatten()

        intersection_value = np.minimum(hist1_value, hist2_value).sum()
        total_pixels_value = hist1_value.sum() + hist2_value.sum()
        similarity_score_value = intersection_value / total_pixels_value if
total_pixels_value != 0 else 0
```

```
        combined_similarity_score = (score_ssim + (1 - intensity_diff/255) +
(1 - sobel_diff/255) + similarity_score_value) / 4
        similarity_scores.append((frame_filenames[i], frame_filenames[i + 1],
combined_similarity_score))

        plt.figure()
        plt.plot(hist1_value, color='b', label='Value (Brightness)')
        plt.title(f'Value Histogram of {frame_filenames[i]}')
        plt.xlabel('Bins')
        plt.ylabel('Frequency')
        plt.legend()

        histogram_filename = os.path.join(histograms_folder,
f'histogram_{frame_filenames[i]}.png')
        plt.savefig(histogram_filename)
        plt.close()

        print(f"Saved histogram for {frame_filenames[i]} and combined
similarity score: {combined_similarity_score}")

    with open(similarity_scores_file, 'w') as f:
        for frame1, frame2, score in similarity_scores:
            f.write(f'{frame1} - {frame2}: {score}\n')

    print("All similarity calculations done, and similarity scores stored.")
    return similarity_scores_file

# Task 5: Mark Scene Cuts based on Similarity Scores (Hard and Soft Cuts)
def detect_scene_cuts_from_similarity(similarity_scores_file, input_folder,
scene_cut_folder='scene_cut_frames', hard_threshold=0.6, soft_threshold=0.7):
    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())

        # Hard Cut Detection
        if score < hard_threshold:
            print(f"Hard Scene cut detected between {frame1} and {frame2} with
similarity score: {score}")
```

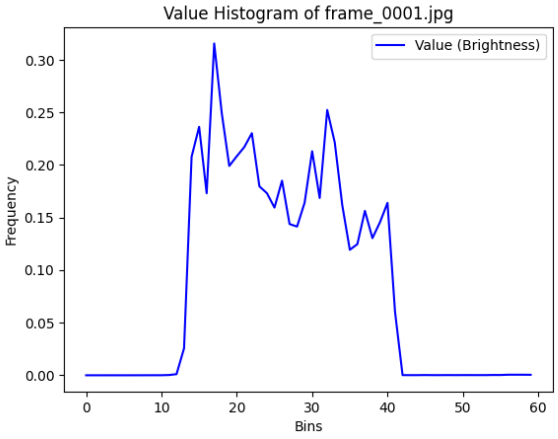
```
            frame1_path = os.path.join(input_folder, frame1.strip())
            frame2_path = os.path.join(input_folder, frame2.strip())

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

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

            cut_detected.append((frame1.strip(), frame2.strip(), "Hard"))

        # Soft Cut Detection
        elif score < soft_threshold:
            print(f"Soft Scene transition detected between {frame1} and
{frame2} with similarity score: {score}")

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

            cut_detected.append((frame1.strip(), frame2.strip(), "Soft"))

    return cut_detected

# Task 6: Result Visualization (Scene Cut Frames)
def visualize_scene_cuts(scene_cut_folder):
    for frame_filename in sorted(os.listdir(scene_cut_folder)):
        frame_path = os.path.join(scene_cut_folder, frame_filename)
        frame = cv2.imread(frame_path)
        cv2.imshow(f'Scene Cut - {frame_filename}', frame)
        cv2.waitKey(0)
    cv2.destroyAllWindows()

# Pipeline execution
video_path = "strawberry.mp4"
frame_folder = extract_frames(video_path)

# HSV frame conversion
hsv_folder = convert_frames_to_hsv(frame_folder)
```

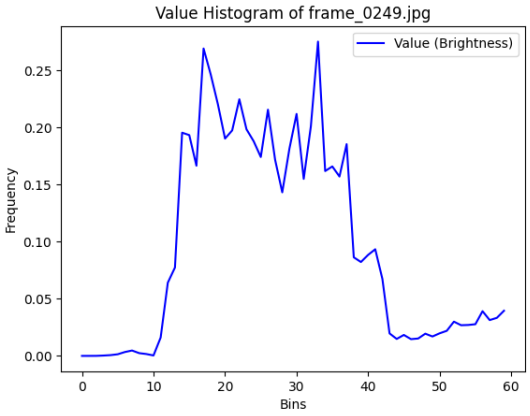
```
# Spatio-temporal segmentation and object tracking (foreground/background)
tracking_folder = background_subtraction_tracking(frame_folder)

# Compute similarity scores using Sobel edges, SSIM, pixel intensity, and
histogram comparisons
similarity_scores_file = compute_similarity_scores(hsv_folder)

# Detect scene cuts and save the detected frames (hard and soft cuts)
scene_cuts = detect_scene_cuts_from_similarity(similarity_scores_file,
frame_folder)

# Visualize scene cut frames
#isualize_scene_cuts('scene_cut_frames')

#print("Processing complete.")
```

6. Result and discussion

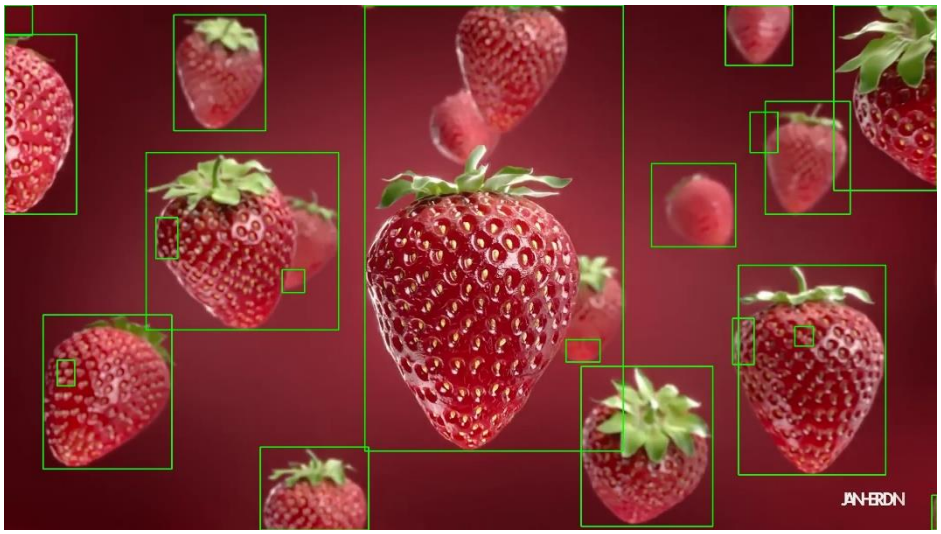


Original frame **HSV converted frame**





First frame histogram and last frame histogram



Object detection in each frame

TRANSITION OF FRAMES IN HARD CUT SCENE

TRANSITION OF FRAMES IN SOFT CUT SCENE

7. Conclusion

- **Summary of Work:**

The task involved importing a video, extracting frames, and performing spatio-temporal segmentation using Sobel edge detection. Scene cuts were identified based on structural similarity index (SSIM), pixel intensity differences, and histogram comparisons. Object tracking was achieved by applying color thresholding in the HSV color space, followed by bounding box detection around tracked objects. After converting frames to the HSV color space, foreground/background segmentation and object tracking are performed using background subtraction. SSIM scores are then computed for each frame pair, incorporating additional metrics like intensity differences and Sobel edge detection.

The code effectively implements a comprehensive scene cut detection framework, classifying hard and soft cuts in video frames using multiple image processing techniques. The classification is based on two thresholds: **hard cut for similarity scores below 0.6** and **soft cut for scores between 0.6 and 0.7.** There are totally 61 hardcut scene and 71 soft cut scenes.

The processed outputs are stored in separate folders, with hard and soft scene transitions saved in the `**scenecut_frames**` folder. This method efficiently distinguishes between abrupt and gradual transitions, providing clear insights into scene changes and object tracking within video sequences.

- **Key Takeaways:**

1. SSIM was effective at detecting scene changes by comparing structural similarities between successive frames.
2. Histogram-based approaches improved SSIM by detecting color changes in the video, particularly during gentle transitions.
3. Sobel edge detection was effective at segmenting objects based on their borders, making it easier to track their movements between frames.

4. Accurate scene cut detection requires the integration of numerous techniques such as SSIM, intensity difference, and histogram comparison.