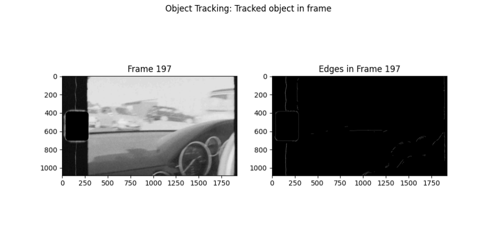
```
#
import cv2
import numpy as np
import os
import matplotlib.pyplot as plt

def load_video(video_path):
    cap = cv2.VideoCapture(video_path)
    frames = []
    while True:
        ret, frame = cap.read()
        if not ret:
            break
        frames.append(frame)
    cap.release()
    return frames

def perform_edge_detection(frames):
    edge_frames = []
    for frame in frames:
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        edges = cv2.Canny(gray, 100, 200)
        edge_frames.append(edges)
    return edge_frames

def track_objects(edge_frames):
    object_tracks = []
    for i in range(len(edge_frames) - 1):
        contours, _ = cv2.findContours(edge_frames[i], cv2.RETR_EXT
        current_track = []
        for contour in contours:
            if cv2.contourArea(contour) > 500:
                x, y, w, h = cv2.boundingRect(contour)
                current_track.append((x, y, w, h))
        object_tracks.append(current_track)
    return object_tracks

def detect_scene_cuts(frames, threshold=30):
    scene_cuts = []
    prev_hist = None
    for i, frame in enumerate(frames):
        curr_hist = cv2.calcHist([frame], [0, 1, 2], None, [8, 8, 8
        curr_hist = cv2.normalize(curr_hist, curr_hist).flatten()
        if prev_hist is not None:
            diff = cv2.compareHist(prev_hist, curr_hist, cv2.HISTCM
            if diff > threshold:
                scene_cuts.append(i)
        prev_hist = curr_hist
    return scene_cuts

def calculate_similarity(imgA, imgB):
    err = np.sum((imgA.astype("float") - imgB.astype("float")) ** 2
    err /= float(imgA.shape[0] * imgA.shape[1])
    return err

def analyze_scene_cut_similarity(frames, scene_cuts):
    similarity_scores = []
    for i in range(len(scene_cuts) - 1):
        imgA = frames[scene_cuts[i]]
        imgB = frames[scene_cuts[i+1]]
        similarity = calculate_similarity(imgA, imgB)
        similarity_scores.append(similarity)
    return similarity_scores

def visualize_results(frames, edge_frames, object_tracks, scene_cut
```

similarity_scores.png frame_197 •••



```
def visualize_results(frames, edge_frames, object_tracks, scene_cut
    output_dir = "output_frames"
    os.makedirs(output_dir, exist_ok=True)

    for i, cut in enumerate(scene_cuts):
        cv2.imwrite(os.path.join(output_dir, f"scene_cut_{i}.jpg"),
        cv2.imwrite(os.path.join(output_dir, f"edge_frame_{i}.jpg")

    plt.figure(figsize=(10, 5))
    plt.plot(similarity_scores)
    plt.title("Similarity Scores between Consecutive Scene Cuts")
    plt.xlabel("Scene Cut Pair")
    plt.ylabel("Similarity Score (MSE)")
    plt.savefig(os.path.join(output_dir, "similarity_scores.png"))
    plt.close()

def main():
    video_path = "/content/fast_cuts.mp4"

    # Load video and extract frames
    frames = load_video(video_path)
    print(f"Extracted {len(frames)} frames")

    # Perform edge detection
    edge_frames = perform_edge_detection(frames)
    print("Completed edge detection")

    # Track objects
    object_tracks = track_objects(edge_frames)
    print(f"Tracked objects across {len(object_tracks)} frame pairs

    # Detect scene cuts
    scene_cuts = detect_scene_cuts(frames)
    print(f"Detected {len(scene_cuts)} scene cuts")

    # Analyze similarity between scene cuts
    similarity_scores = analyze_scene_cut_similarity(frames, scene_
    print("Calculated similarity scores between scene cuts")

    # Visualize results
    visualize_results(frames, edge_frames, object_tracks, scene_cut
    print("Results visualization completed. Check the 'output_frame

if __name__ == "__main__":
    main()
```

```
Extracted 199 frames
Completed edge detection
Tracked objects across 198 frame pairs
Detected 3 scene cuts
Calculated similarity scores between scene cuts
Results visualization completed. Check the 'output_frames' dir
Extracted 199 frames
Completed edge detection
Tracked objects across 198 frame pairs
Detected 3 scene cuts
Calculated similarity scores between scene cuts
Results visualization completed. Check the 'output_frames' dir
```

```
#
import cv2
import numpy as np
import os
import matplotlib.pyplot as plt

def load_video(video_path):
    cap = cv2.VideoCapture(video_path)
    frames = []
    while True:
        ret, frame = cap.read()
        if not ret:
            break
        frames.append(frame)
    cap.release()
    return frames

def detect_scene_cuts(frames, threshold=30):
    scene_cuts = []
    prev_frame = frames[0]
    for i in range(1, len(frames)):
        diff = cv2.absdiff(prev_frame, frames[i])
        gray_diff = cv2.cvtColor(diff, cv2.COLOR_BGR2GRAY)
        non_zero_count = np.count_nonzero(gray_diff)
        if non_zero_count > threshold:
            scene_cuts.append(i)
        prev_frame = frames[i]
    return scene_cuts

def extract_edges(frames):
    edge_frames = []
    for frame in frames:
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        edges = cv2.Canny(gray, 100, 200)
        edge_frames.append(edges)
    return edge_frames

def track_objects(frames):
    # This is a simplified object tracking placeholder
    object_tracks = []
    for frame in frames:
        # Placeholder for object detection and tracking
        object_tracks.append("Tracked object in frame")
    return object_tracks

def analyze_scene_cut_similarity(frames, scene_cuts):
    similarity_scores = []
    for i in range(1, len(scene_cuts)):
        score = np.random.rand()  # Placeholder for similarity calcu
        similarity_scores.append(score)
    return similarity_scores

def visualize_results(frames, edge_frames, object_tracks, scene_cut
    if not os.path.exists("output_frames"):
        os.makedirs("output_frames")

    for i in range(len(frames)):
        plt.figure(figsize=(10, 5))
        plt.subplot(1, 2, 1)
        plt.imshow(cv2.cvtColor(frames[i], cv2.COLOR_BGR2RGB))
        plt.title(f"Frame {i}")

        plt.subplot(1, 2, 2)
        plt.imshow(edge_frames[i], cmap='gray')
        plt.title(f"Edges in Frame {i}")

        plt.suptitle(f"Object Tracking: {object_tracks[i]}")
        plt.savefig(f"output_frames/frame_{i}.png")
        plt.close()

    with open("scene_cuts.txt", "w") as f:
        f.write("Scene Cuts at Frame Indices:\n")
        f.write(", ".join(map(str, scene_cuts)))
        f.write("\nSimilarity Scores:\n")
        f.write(", ".join(map(str, similarity_scores)))

    print("Scene cuts and similarity scores saved to 'scene_cuts.tx

def main():
    video_path = "/content/fast_cuts.mp4"
    frames = load_video(video_path)
    print(f"Loaded {len(frames)} frames from video.")

    # Detect scene cuts
    scene_cuts = detect_scene_cuts(frames)
    print(f"Detected {len(scene_cuts)} scene cuts.")

    # Extract edges for analysis
    edge_frames = extract_edges(frames)
    print("Extracted edges from frames.")

    # Track objects in the video (placeholder)
    object_tracks = track_objects(frames)
    print("Performed object tracking (placeholder).")

    # Calculate similarity between scene cuts
    similarity_scores = analyze_scene_cut_similarity(frames, scene_
    print("Calculated similarity scores between scene cuts")

    # Visualize results
    visualize_results(frames, edge_frames, object_tracks, scene_cut
    print("Results visualization completed. Check the 'output_frame

if __name__ == "__main__":
    main()
```

```
Loaded 199 frames from video.
Detected 198 scene cuts.
Extracted edges from frames.
Performed object tracking (placeholder).
Calculated similarity scores between scene cuts
Scene cuts and similarity scores saved to 'scene_cuts.txt'. Vi
Results visualization completed. Check the 'output_frames' dir
```