```
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_EXTERNAL, cv2.CHAIN_APPR
        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], [0, 256, 0, 256, 0
        curr_hist = cv2.normalize(curr_hist, curr_hist).flatten()
        if prev_hist is not None:
            diff = cv2.compareHist(prev_hist, curr_hist, cv2.HISTCMP_CHISQR)
            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_cuts, similarity_scores)
    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"), frames[cut])
        cv2.imwrite(os.path.join(output_dir, f"edge_frame_{i}.jpg"), edge_frames[cut])

    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)")
```

scene_cut_12.jpg scene_cut_2.jpg scen •••

```
        plt.savefig(os.path.join(output_dir, "similarity_scores.png"))
        plt.close()

    def main():
        video_path = "/content/Untitled video - Made with Clipchamp (1).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_cuts)
        print("Calculated similarity scores between scene cuts")

        # Visualize results
        visualize_results(frames, edge_frames, object_tracks, scene_cuts, similarity_scores)
        print("Results visualization completed. Check the 'output_frames' directory for save

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

```
Extracted 562 frames
Completed edge detection
Tracked objects across 561 frame pairs
Detected 13 scene cuts
Calculated similarity scores between scene cuts
Results visualization completed. Check the 'output_frames' directory for saved ima
```

Start coding or generate with AI.