Program-4:

```
import torch
import torchvision
import cv2
import torchvision.transforms.functional as F
import numpy as np
from google.colab.patches import cv2_imshow  # Import cv2_imshow for
Colab
import time

# Load the pre-trained Faster R-CNN model
device = torch.device('cuda') if torch.cuda.is_available() else
torch.device('cpu')
model =
torchvision.models.detection.fasterrcnn_resnet50_fpn(pretrained=True)
model.eval().to(device)

# COCO labels (you already have them)
COCO_INSTANCE_CATEGORY_NAMES = [
    '__background__', 'person', 'bicycle', 'car', 'motorcycle',
'airplane', 'bus',
    'train', 'truck', 'boat', 'traffic light', 'fire hydrant', 'N/A',
'stop sign',
    'parking meter', 'bench', 'bird', 'cat', 'dog', 'horse', 'sheep',
'cow',
    'elephant', 'bear', 'zebra', 'giraffe', 'N/A', 'backpack',
'umbrella', 'N/A', 'N/A',
    'handbag', 'tie', 'suitcase', 'frisbee', 'skis', 'snowboard',
'sports ball',
    'kite', 'baseball bat', 'baseball glove', 'skateboard',
'surfboard', 'tennis racket',
    'bottle', 'N/A', 'wine glass', 'cup', 'fork', 'knife', 'spoon',
'bowl',
    'banana', 'apple', 'sandwich', 'orange', 'broccoli', 'carrot', 'hot
dog', 'pizza',
    'donut', 'cake', 'chair', 'couch', 'potted plant', 'bed', 'dining
table',
    'N/A', 'N/A', 'toilet', 'N/A', 'tv', 'laptop', 'mouse', 'remote',
'keyboard', 'cell phone','microwave', 'oven', 'toaster', 'sink',
'refrigerator', 'N/A', 'book',
    'clock', 'vase', 'scissors', 'teddy bear', 'hair drier',
'toothbrush'
]

# Function to perform object detection on each frame of a video
def detect_objects_in_video(video_path, confidence_threshold=0.5):
```

```
    # Open the video stream (video file or webcam)
    cap = cv2.VideoCapture(video_path)  # For webcam, pass 0 or the
index of your webcam
    if not cap.isOpened():
        print("Error: Could not open video stream.")
        return

    # Process the video frame by frame
    while True:
        ret, frame = cap.read()
        if not ret:
            break  # End of video

        # Convert frame to RGB for the model
        frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        image_tensor = F.to_tensor(frame_rgb).unsqueeze(0).to(device)

        # Perform inference
        with torch.no_grad():
            predictions = model(image_tensor)

        # Extract bounding boxes, labels, and scores
        boxes = predictions[0]['boxes'].cpu().numpy()
        labels = predictions[0]['labels'].cpu().numpy()
        scores = predictions[0]['scores'].cpu().numpy()

        # Filter detections based on the confidence threshold
        for i, box in enumerate(boxes):
            if scores[i] >= confidence_threshold:
                label = COCO_INSTANCE_CATEGORY_NAMES[labels[i]]
                score = scores[i]
                start_point = (int(box[0]), int(box[1]))
                end_point = (int(box[2]), int(box[3]))
                # Draw bounding box and label
                cv2.rectangle(frame, start_point, end_point, (0, 255,
0), 3)
                cv2.putText(frame, f"{label}: {score:.2f}",
start_point, cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255, 255, 0), 3)

        # Display the frame with detections using cv2_imshow for Colab
        cv2_imshow(frame)  # Display using cv2_imshow instead of
cv2.imshow()

        # Wait for 1ms and check for 'q' key to exit (in case of video
file)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
```

```
    # Release the video capture object and close the display window
    cap.release()
    cv2.destroyAllWindows()

# Example usage for video file or webcam
video_path = "/content/vecteezy_cheetah-running-in-the-
savanna_52003696.mov"  # Replace this with your video path or use 0 for
webcam
detect_objects_in_video(video_path, confidence_threshold=0.5)
```

Modifications in the New Code:

1. **Input Source Changed**:
   - Replaced static image input with video input using `cv2.VideoCapture(video_path)` to handle video files or webcam streams.
2. **Frame-by-Frame Processing**:
   - Added a `while` loop to read and process video frames iteratively using `cap.read()`.
3. **RGB Conversion for Frames**:
   - Each frame is converted to RGB format using `cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)` for compatibility with the model.
4. **Dynamic Frame Display**:
   - Used `cv2_imshow(frame)` to display processed frames continuously in real-time.
5. **Exit Condition**:
   - Incorporated a `cv2.waitKey(1) & 0xFF == ord('q')` condition to allow the user to stop processing before the video ends.
6. **Resource Management**:
   - Added `cap.release()` and `cv2.destroyAllWindows()` to release video resources and close display windows after processing.