Documentation for Object and Sub-Object Detection System

1. Introduction

This system is designed to detect objects and their associated sub-objects in a hierarchical structure using a computer vision approach. It processes video input, identifies objects and sub-objects, and generates JSON outputs detailing detected entities and their relationships. Additionally, it extracts and saves images of sub-objects for further analysis. The system ensures modularity, extensibility, and real-time performance on a CPU.

2. Features

- **Object and Sub-Object Detection**: Identifies objects like "Person," "Car," etc., and sub-objects like "Helmet," "Tire," etc.
- **Hierarchical JSON Output**: Provides structured outputs linking sub-objects to their parent objects.
- **Sub-Object Image Retrieval**: Allows retrieval and saving of cropped images for specific sub-objects.
- **Real-Time Processing**: Optimized for 10-30 frames per second (FPS) on CPU.
- **Modular Design**: Easily adaptable to new object-sub-object pairs.

3. System Requirements

- **Python Version**: Python 3.7 or later
- **Dependencies**: Listed in `requirements.txt`
- **Supported Frameworks**: Ultralytics YOLO, OpenCV

Installation

1. Clone the repository or download the source code.
2. `git clone <repository_url>`
3. `cd <repository_directory>`
4. Install required dependencies:
5. `pip install -r requirements.txt`

4. File Structure

- **`main.py`**: Entry point for processing video input.
- **`model.py`**: Defines the model loading and inference logic.
- **`utils.py`**: Contains helper functions for JSON generation, benchmarking, and image saving.
- **`requirements.txt`**: Specifies required Python libraries.

- **`outputs/`**: Directory where JSON and cropped images are saved.

5. Usage

Running the System

1. Ensure the `outputs/` directory exists or create it manually:
2. `mkdir outputs`
3. Run the script with the video input:
4. `python main.py --video path/to/your/video.mp4`
5. The system will process the video, display detection outputs, and save results in the `outputs/` directory.

Sample JSON Output

```
{
  "object": "Car",
  "id": 1,
  "bbox": [100, 150, 200, 250],
  "subobject": {
    "object": "Tire",
    "id": 1,
    "bbox": [110, 160, 120, 170]
  }
}
```

Performance Benchmark

- **Inference Speed**: ~9-11 FPS on CPU
- **Preprocessing and Postprocessing Times**: Averaged at ~3 ms each

6. Code Highlights

`main.py`

Processes video frames, performs detection, and saves JSON outputs and cropped images. Key logic:

```
os.makedirs('outputs', exist_ok=True)
results = model.predict(frame)
json_output = create_json_output(results)
with open('outputs/detections.json', 'w') as f:
    json.dump(json_output, f, indent=4)
```

`model.py`

Loads the YOLO model and performs inference:

```
model = YOLO('yolov8n.pt')
def predict(frame):
    return model(frame)
```

`utils.py`

Generates JSON and saves cropped images:

```
def save_cropped_image(frame, bbox, filename):
    crop = frame[bbox[1]:bbox[3], bbox[0]:bbox[2]]
    cv2.imwrite(f'outputs/{filename}', crop)
```

7. Limitations and Assumptions

- **Assumptions**: The system assumes clear visibility of objects and sub-objects.
- **Limitations**: Performance may degrade with overlapping or occluded objects.

8. Extending the System

To add new object-sub-object pairs:

1. Modify `model.py` to include new detection logic.
2. Update `utils.py` for specific sub-object associations.

9. Conclusion

This hierarchical object and sub-object detection system is a robust and modular solution for real-time video analysis. Its design ensures adaptability to various detection scenarios, making it suitable for deployment on edge devices.