ASSESSMENT Report

Proposed name : Pixeleye

Assesment Given By : Safetywhat

Submitted By: Vishesh Yadav

Enrollment No: 12322114

Degree: MCA hons AI & ML

Project Name: Pixeleye

Github Link

https://github.com/vishesh9131/SafetyWhat?tab=readme-ov-file#optimization-strategies

Table of Contents

Approach : Subobject Detection and IoU Logic

This Assessment uses the YOLOv5 model to detect both main objects and their subobjects within video frames. The model is configured with specific confidence and IoU thresholds to ensure accurate detections.

Subobject Detection

- Process: After detecting objects, the system checks for subobjects by analyzing the spatial relationship between detected objects.
- Rules: For example, glasses should be positioned on the upper part of a face, and a watch should be centered on a wrist. These rules help confirm valid subobject detections.
- Validation: The system uses IoU to ensure that subobjects are correctly positioned relative to the main object.

Intersection over Union (IoU)

- Definition: IoU measures the overlap between two bounding boxes, calculated as the area of intersection divided by the area of union.
- Usage: IoU helps verify that a subobject is sufficiently overlapping with its main object, ensuring accurate detection.
- Thresholds: Different IoU thresholds are set for various subobjects to handle different detection scenarios.

This approach ensures reliable detection of subobjects by combining YOLOv5's capabilities with custom rules and IoU validation.

Installation

Environment Setup

1. Create a Conda Environment

Run the following commands to create a Conda environment named `safewt-asses` with Python 3.13:

- `conda create -n safewt-asses python=3.13 -y`
- `conda activate safewt-asses`

2. Install Dependencies

Install the required dependencies using pip:

- `pip install -r requirements.txt`

3. Run the Code

- `python -u "main.py" --video locat/test_video.mp4`

Configurations:

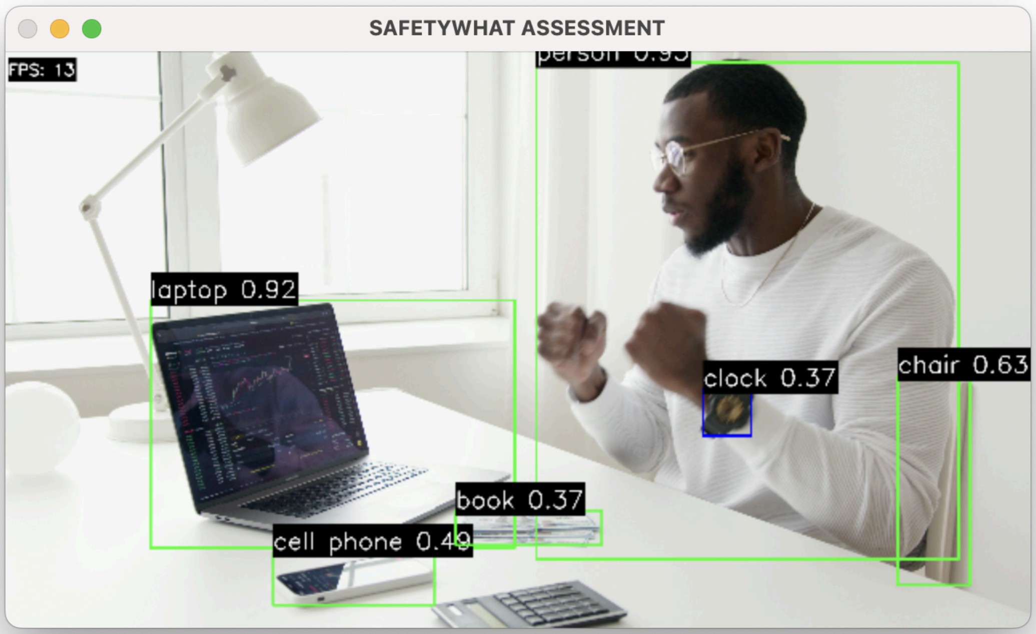
- `--webcam`: Use webcam for real-time detection.
- `--video`: Path to input video (default: `v.mp4`).
- `--conf-threshold`: Confidence threshold for detections (default: `0.3`).
- `--iou-threshold`: IoU threshold for NMS (default: `0.3`).
- `--max-det`: Max detections per image (default: `100`).
- `--img-size`: Input image size (default: `640`).
- `--frame-skip`: Skip frames for increased FPS (default: `1`).
- `--model`: YOLOv5 model type (`yolov5n`, `yolov5s`, `yolov5m`, `yolov5l`, `yolov5x`).

default model is `yolov5s`

Test Videos: You can also download test videos from this drive_link.

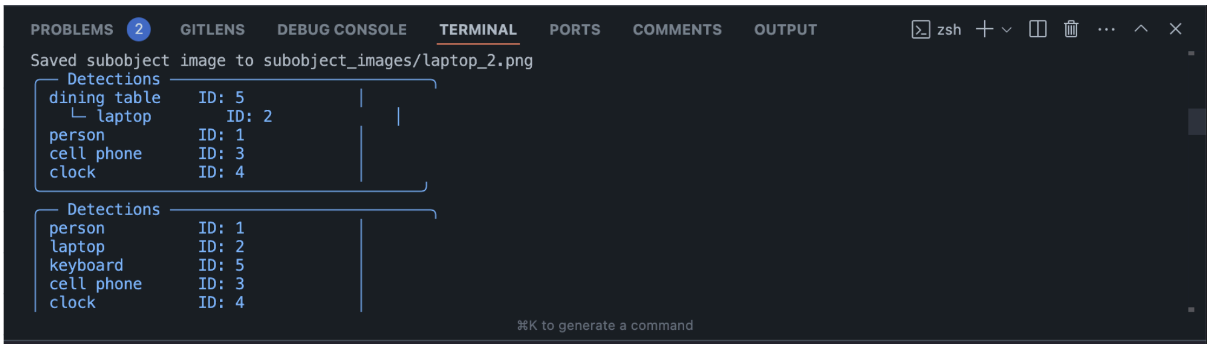
4. Output you will get

- **Interface**: Displays the processed video with detection overlays in a user interface.



Note: This Output used yolov5s model (FPS: 13-25)

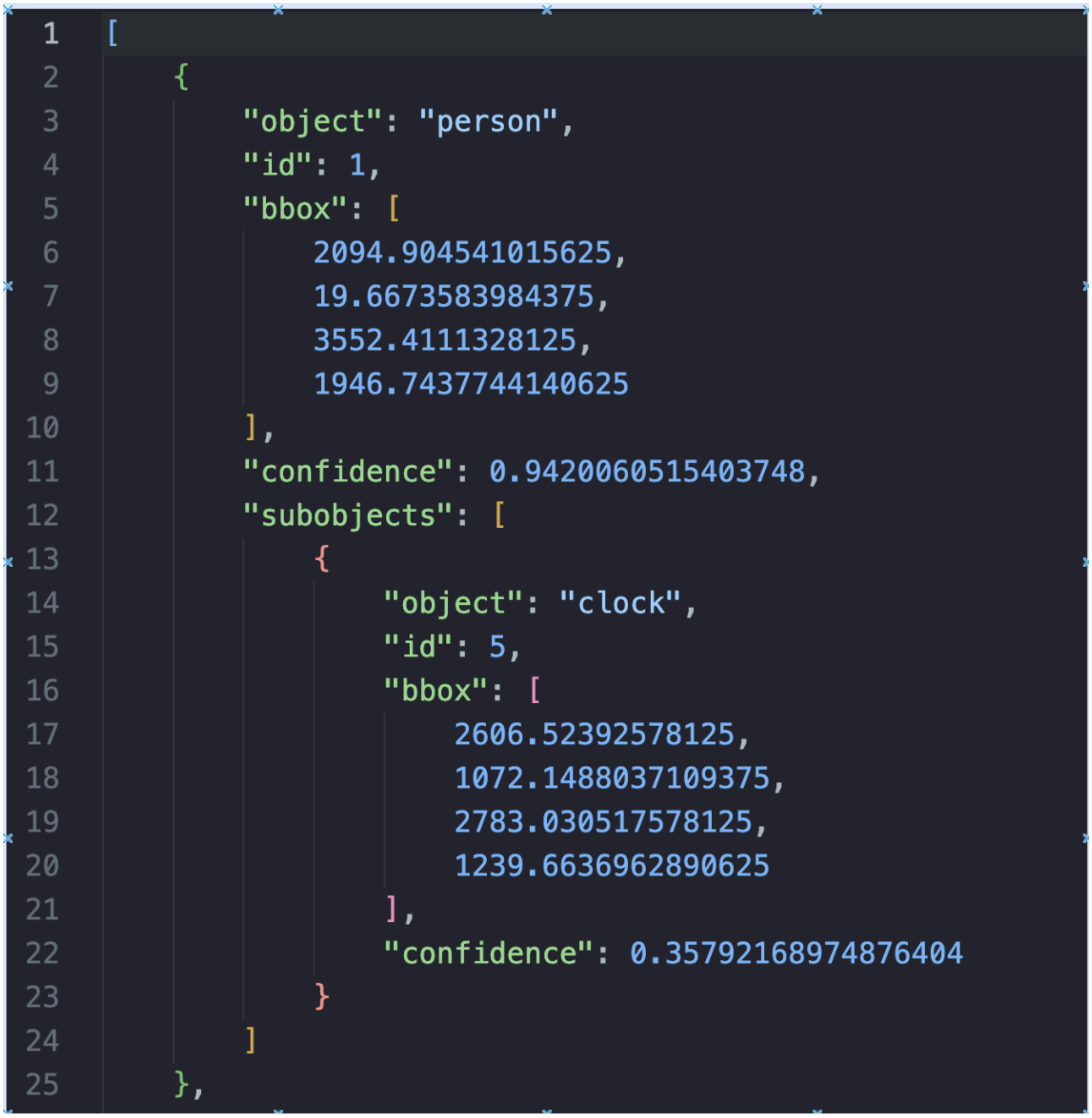
- **final_output.json**: Stores the detection results, including objects and subobjects, in JSON format for further analysis.



```
Saved subobject image to subobject_images/laptop_2.png
┌─ Detections ──────────────────────────────┐
│ dining table     ID: 5               │
│    └─ laptop        ID: 2                │
│ person           ID: 1               │
│ cell phone       ID: 3               │
│ clock            ID: 4               │
└───────────────────────────────────────────┘
┌─ Detections ──────────────────────────────┐
│ person           ID: 1               │
│ laptop           ID: 2               │
│ keyboard         ID: 5               │
│ cell phone       ID: 3               │
│ clock            ID: 4               │
```

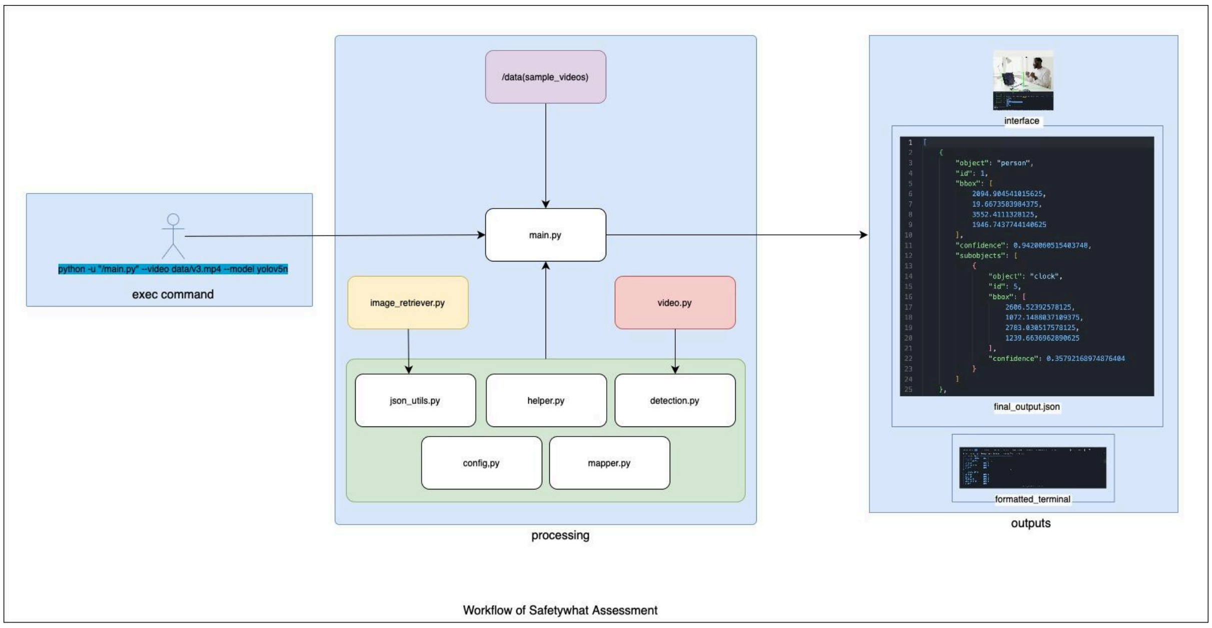
Note: (After running the code you will get this output in **final_output.json**)

- **formatted_terminal**: Outputs formatted detection information to the terminal for real-time monitoring.



```
[
    {
        "object": "person",
        "id": 1,
        "bbox": [
            2094.904541015625,
            19.6673583984375,
            3552.4111328125,
            1946.7437744140625
        ],
        "confidence": 0.9420060515403748,
        "subobjects": [
            {
                "object": "clock",
                "id": 5,
                "bbox": [
                    2606.52392578125,
                    1072.1488037109375,
                    2783.030517578125,
                    1239.6636962890625
                ],
                "confidence": 0.35792168974876404
            }
        ]
    },
```

Workflow of SafetyWhat Assessment (pixeleye)



```
[
    {
        "object": "person",
        "id": 1,
        "bbox": [
            2094.904541015625,
            19.6673583984375,
            3552.4111328125,
            1946.7437744140625
        ],
        "confidence": 0.9420060515403748,
        "subobjects": [
            {
                "object": "clock",
                "id": 5,
                "bbox": [
                    2606.52392578125,
                    1072.1488037109375,
                    2783.030517578125,
                    1239.6636962890625
                ],
                "confidence": 0.35792168974876404
            }
        ]
    },
```

This diagram illustrates the workflow of the PixEye object detection system, highlighting the interaction between various components and the flow of data from input to output.

Execution Command

- **User Input**: The process begins with the user executing a command in the terminal: `python -u "main.py" --video data/v3.mp4 --model yolov5n`
- This command specifies the video file to be processed and the model to be used for detection.

Main Processing

- **main.py**: Acts as the central hub, orchestrating the workflow. It initializes the model, processes video frames, and coordinates with other modules.

Data Input

- **/data (sample_videos)**: Contains the video files to be processed. The specified video is loaded and passed to `main.py`.

Processing Modules

- **image_retriever.py**: Handles saving cropped images of detected subobjects.
- **json_utils.py**: Formats and saves detection results in JSON format.
- **helper.py**: Sets up the environment and displays the application banner.
- **video.py**: Manages video capture, FPS calculation, and drawing detections on frames.
- **detection.py**: Initializes the YOLOv5 model and processes detections.
- **config.py**: Parses command-line arguments for configuration.
- **mapper.py**: Provides mappings for subobject detection.

See it in action

For complete video visit : https://youtu.be/pHj8i_UcXSs

green boundry says object and blue says its respective subobject.

Benchmarking Report

Introduction

This section presents the benchmarking results of the YOLOv5 models (YOLOv5n, YOLOv5s, and YOLOv5m) on various video files. The primary focus is on the inference speed, measured in frames per second (FPS), across different model configurations.

System Architecture

- **Hardware**: Benchmarks were conducted on a CPU-based system with an M2 Apple Chip on Macbook M2 AIR machine
- **Software**: Senoma OS.

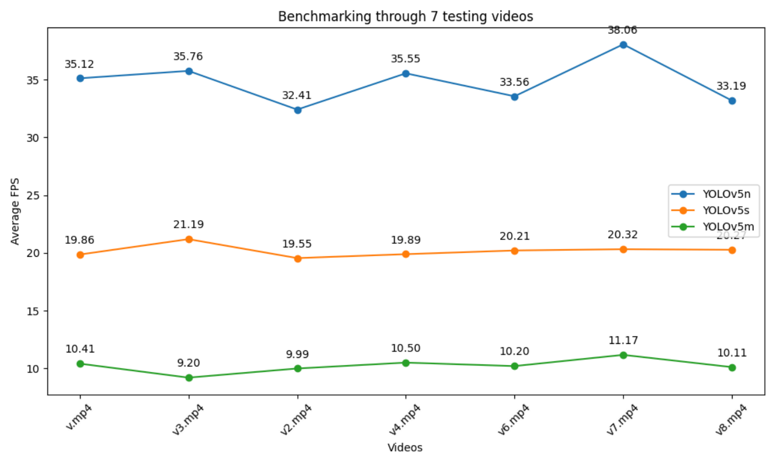
Inference Speed Results

Video	YOLOv5n FPS	YOLOv5s FPS	YOLOv5m FPS
v.mp4	35.12	19.86	10.41
v3.mp4	35.76	21.19	9.20
v2.mp4	32.41	19.55	9.99
v4.mp4	35.55	19.89	10.50
v6.mp4	33.56	20.21	10.20
v7.mp4	38.06	20.32	11.17
v8.mp4	33.19	20.27	10.11

Optimization Strategies

- **Model Pruning**: Reduce parameters and layers.
- **Quantization**: Convert weights to integer.
- **Batch Processing**: Process multiple frames simultaneously.
- **Hardware Acceleration**: Use GPUs or TPUs (in this project used cpu as per requirement)
- **Efficient Data Loading**: Optimize data loading and preprocessing.

Benchmark



System Design and Implementation

Constraints

1. Library and Framework Usage:
    - The system leverages PyTorch and OpenCV for computer vision tasks, as seen in the `requirements.txt` file, which includes `torch`, `torchvision`, and `opencv-python` (startLine: 1, endLine: 3). These libraries provide robust support for deep learning and image processing, making them ideal for implementing YOLO models.
2. CPU-Based Execution:
    - The system is designed to run on a CPU, ensuring it can be deployed on edge devices. This is reflected in the `README.md` where the system architecture specifies a CPU-based setup (startLine: 138, endLine: 140). The use of the `cpuonly` package in the environment setup further emphasizes this constraint.
3. Real-Time Inference Speed:
    - The system achieves real-time inference speeds between 10–30 FPS, as demonstrated in the benchmarking results (startLine: 1, endLine: 230 in `benchmark_results.txt`). Models like YOLOv5n and YOLOv5s consistently meet this requirement, ensuring the system's performance is suitable for real-time applications.

Notes for Success

1. Code Quality:
    - The codebase is structured to be clean and modular, with each component handling specific tasks. For instance, `detection.py` is responsible for initializing the YOLOv5 model and processing detections (startLine: 122, endLine: 122 in `README.md`), while `video.py` manages video capture and FPS calculation (startLine: 121, endLine: 121 in `README.md`).
2. Accuracy and Performance:
    - The system balances accuracy and performance by using different YOLOv5 model variants. The choice of model can be configured via command-line arguments, allowing users to select the model that best fits their needs (startLine: 75, endLine: 75 in `README.md`).
3. Handling Edge Cases:
    - The system is designed to handle occlusion and overlapping objects by leveraging YOLO's inherent capabilities in object detection. The confidence and

IoU thresholds can be adjusted to fine-tune detection sensitivity (startLine: 70, endLine: 71 in `README.md`).

4. Assumptions and Limitations:
    - The system assumes that the input videos are of sufficient quality for object detection. Limitations include potential performance degradation on very low-end CPUs and the need for further optimization for specific edge cases like extreme occlusion.

Conclusion

The system is designed to be portable, scalable, and efficient, meeting the constraints and performance requirements outlined. By focusing on modularity and leveraging powerful libraries, it provides a robust solution for real-time object detection on edge devices.

References

- **YOLOv5 Documentation**: YOLOv5 GitHub Repository
- **PyTorch Documentation**: PyTorch Official Website
- **Conda Documentation**: Conda Official Documentation
- **Matplotlib Documentation**: Matplotlib Official Website
- **Seaborn Documentation**: Seaborn Official Website