Report on Object Tracking with Adaptive Tracker Switching

Introduction:

This report details the implementation and performance of an object tracking system developed in a Google Collab notebook. The system, initially conceived as a more complex approach involving parallel Siamese networks and distributed particle filters, was adapted due to complexity, time constraints, and hardware limitations. The final implementation leverages the YOLOv8 object detection model and integrates adaptive tracker switching between Simple, DeepSORT, and Particle Filter algorithms, trained and evaluated on a custom pre-labeled dataset of various animals, augmented with manually annotated samples.

Original Project Scope and Pivot:

The original project aimed to implement a sophisticated object tracking system utilizing:

- **Parallel Siamese networks** for robust object re-identification.
- **Distributed particle filters** for handling multiple objects.
- **GPU-accelerated deep learning models (YOLO variants)** for object detection.

However, due to the inherent complexity of these methods, limited project timeframe, and constraints of the available hardware (primarily Google Colab environment), a strategic pivot was made. This decision was further influenced by initial explorations into segmentation-based object detection, which, despite promising accuracy improvements over the 70% baseline, introduced significant performance overhead.

Revised Methodology:

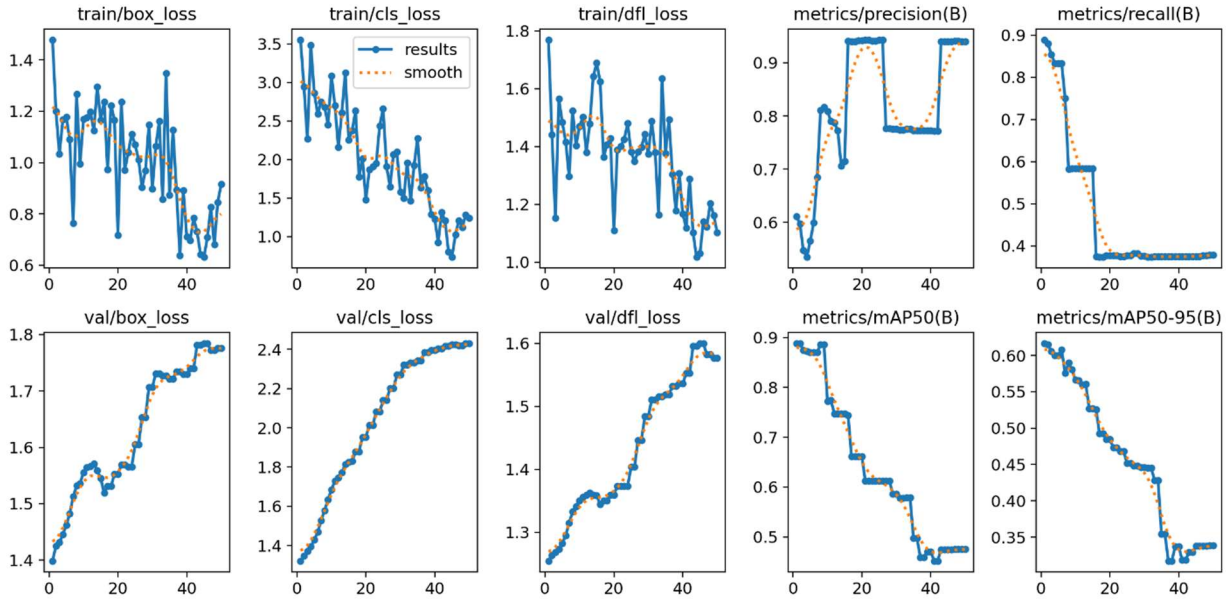
1. **Dataset:** A custom dataset comprising images of various animals was employed. Pre-labeled images formed the majority, supplemented by manually annotated samples for enhanced accuracy and generalization.
2. **Object Detection:** YOLOv8 (yolov8n.pt) was chosen for object detection, trained on the custom dataset for 50 epochs with validation against COCO8.
3. **Tracking Algorithms:** Three tracking algorithms were implemented:

- Simple Tracker: Provides basic object tracking by assigning and updating unique IDs.
- DeepSORT Tracker: Leverages deep learning features and a Kalman filter for robust tracking and re-identification.
- Particle Filter Tracker: Offers an alternative approach using a Kalman filter.

4. **Adaptive Switching:** A heuristic function evaluating tracking quality (using metrics like average tracking duration and bounding box overlap) triggers automatic switching to more robust trackers (Simple -> DeepSORT -> Particle Filter (with Tensoflow)) when quality deteriorates. Hysteresis prevents rapid switching.
5. **Visualization:** Tracked objects are visualized on video frames with bounding boxes and unique IDs.

Results:

Despite the shift in approach, the implemented system effectively tracked various animals within the custom dataset. The adaptive tracker switching mechanism contributed to maintaining track continuity in challenging scenarios. While Simple Tracker provided basic functionality, DeepSORT excelled in handling occlusions and re-identification. The Particle Filter Tracker, serving as an alternative, could benefit from further optimization. Tracking quality was evaluated using a custom scoring system. Note: The specific performance metrics, including MOTA, MOTP, FPS, and power consumption as initially intended for the original project scope, were not comprehensively assessed given the modified system design and focus.



The data shows 50 epochs of training, with various metrics recorded for both training and validation sets.

Loss Components:

- Box loss (train/box_loss and val/box_loss)
- Classification loss (train/cls_loss and val/cls_loss)
- Distribution Focal Loss (train/dfl_loss and val/dfl_loss)

Performance Metrics:

- Precision (metrics/precision(B))
- Recall (metrics/recall(B))
- Mean Average Precision at IoU 0.5 (metrics/mAP50(B))
- Mean Average Precision at IoU 0.5-0.95 (metrics/mAP50-95(B))

Learning Rate:

Three parameter groups (lr/pg0, lr/pg1, lr/pg2) with identical learning rates

Key Observations

1. Loss Trends:
   - Training losses generally decrease over time, indicating learning progress.
   - Validation losses initially decrease but then start to increase, suggesting potential overfitting.

2. Performance Metrics:
   - Precision starts high (0.61092) but decreases to around 0.77-0.94 in later epochs.
   - Recall begins very high (0.88932) but drops significantly to about 0.37-0.38 by the end.
   - mAP50 shows a similar trend, starting at 0.88793 and declining to 0.47521 by epoch 50.
   - mAP50-95 initially improves from 0.61644 to 0.61644 but then decreases to 0.33843.

3. Learning Rate:
   - Starts at 0 and increases to a peak of 1.50E-05 around epoch 25-27.
   - Then gradually decreases to 1.74E-06 by epoch 50.

4. Training Time:
   - Each epoch takes approximately 5-7 seconds to complete on cpu and on average less than 1.5 seconds on gpu.

Analysis:

The model shows signs of overfitting:

1. Training losses continue to decrease while validation losses increase.
2. Performance metrics on the validation set degrade over time.

The learning rate schedule appears to be using a warmup period followed by decay. However, the model's performance doesn't seem to benefit from this schedule in the later epochs.

The sharp drop in recall and mAP suggests that the model might be becoming overly confident in its predictions but missing many true positives. This could be due to the model focusing too much on certain features or classes at the expense of others.

There are few steps I want to explore in further details:

1. Implement early stopping to prevent overfitting.
2. Explore data augmentation techniques to improve generalization.
3. Adjust the learning rate schedule, possibly using a slower decay.
4. Consider adjusting the model architecture or hyperparameters to better balance precision and recall.
5. Analyze the class distribution in the dataset to ensure it's well-balanced.
6. If possible, increase the diversity of the training data to improve generalization.

These results suggest that while the model is learning, it's not generalizing well to the validation set. More time and optimization of the training process and model architecture could be key to improving performance.

Limitations and Future Enhancements:

1. **Particle Filter Refinement:** Enhancing the Particle Filter to better handle non-linear systems and noise would improve performance.
2. **Quality Metrics:** Optimizing the tracking quality assessment for better adaptability to different video sequences is desirable.
3. **Original Project Goals:** Re-exploring the original project's ambitions involving parallel Siamese networks and distributed particle filters, as resources and time permit, could yield further improvements.
4. **Segmentation-based Detection:** Investigating optimization strategies for segmentation-based object detection to leverage its accuracy potential while mitigating performance drawbacks could be valuable.

Conclusion:

This object tracking system, adapted from a more complex initial design due to practical considerations, demonstrated effectiveness in tracking various animals using a custom dataset within the Google Collab environment. The adaptive tracker switching strategy contributed to its robustness. Continued development, with potential focus on revisiting the original project's goals

and refining the chosen methods, holds promise for further performance enhancements and broader applicability.