Deep Learning

Instructor: Prof. E. Fatemizadeh

Sharif University of Technology

Object tracking on Sports Dataset

Authors:

Mahdi Tabatabaei
Student ID: 400101515

Tabatabaeii.Mahdii@gmail.com

Winter 2024

Contents

1 Dataset Preparation and Preprocessing **2**
1.1 SportsMOT 2
1.2 Soccernet 3

2 Object Tracking **3**
2.1 Object Detection 3
2.2 Single Object Tracking 7
2.3 Multiple Object Tracking 11

3 Model Improvement **15**
3.1 Research 15

1 Dataset Preparation and Preprocessing

In this phase, we selected the `SportsMOT` and `Soccernet` datasets for our implementations.

1.1 SportsMOT

This dataset includes videos from football, volleyball, and basketball games. However, for this project, we used only football videos.

1.1.1 Preprocessing

To augment the dataset, we applied frame flipping and brightness adjustment. It is important to note that when flipping frames, all frames in a video must be flipped consistently. However, brightness changes should be applied gradually to maintain smooth transitions across sequential frames.

Based on our analysis, this dataset does not contain any outliers, so no data was removed. We applied the following transformations for normalization:

```
import torchvision.transforms as transforms

train_transform = transforms.Compose([
    transforms.Resize((720, 1280)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])

val_transform = transforms.Compose([
    transforms.Resize((720, 1280)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])
```

1.1.2 Preparation

The dataset is already divided into training, validation, and test sets. Our task is to format the data appropriately for YOLO training. The required file format should be structured as illustrated in Figure 1.

We should create the labels for YOLO in the following format:

$$(x_{center}, y_{center}, width, height)$$

```
x_center = ((x_min + x_max) / 2) / img_width
y_center = ((y_min + y_max) / 2) / img_height
width = (x_max - x_min) / img_width
height = (y_max - y_min) / img_height
```

After that, we should create our '`.yaml`' to give it as input of our YOLO model.

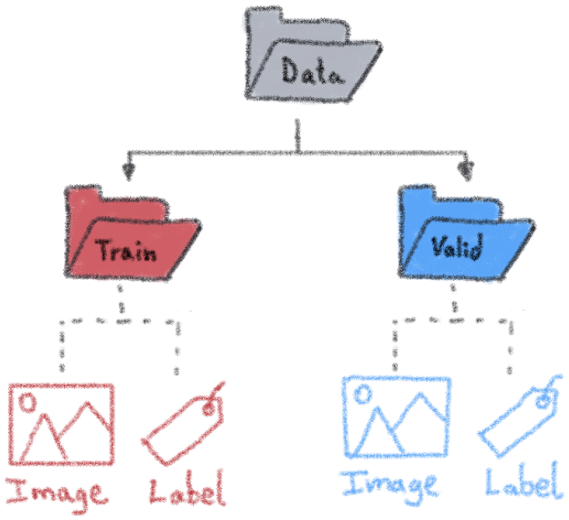

Figure 1: Directory Structure to train YOLO model

1.2 Soccernet

This dataset contains a large amount of data, and no data augmentation was performed. Additionally, it provides ground truth annotations for both the ball and referees. I applied the same preprocessing and preparation steps to this dataset as we did for the previous one.

2 Object Tracking

2.1 Object Detection

2.1.1 YOLO Model

For object detection, we utilized the YOLO model. YOLO (You Only Look Once) is well-suited for this task due to its efficiency and accuracy in object detection. Compared to other methods, we choose YOLO for following reasons:

- **Speed:** YOLO processes images in real-time, making it highly efficient for applications requiring fast detection. It is also quicker and more suitable for 25 fps.
- **Single-Pass Detection:** Unlike region-based approaches, YOLO predicts bounding boxes and class probabilities in a single forward pass.
- **High Accuracy:** With improved architectures such as YOLOv5 and YOLOv8, the model achieves state-of-the-art accuracy.
- **Generalization:** YOLO performs well across various datasets without requiring excessive fine-tuning.
- **Tracking algorithms:** Many tracking models (like DeepSORT, ByteTrack) are optimized for YOLO.

2.1.2 Implementation

The following steps were taken to implement YOLO for object detection:

- Dataset formatting to meet YOLO's required annotation format.
- Model selection and hyperparameter tuning for optimal performance.
- Training using pre-trained YOLO weights and fine-tuning on the dataset.
- Post-processing predictions to filter detected objects and improve precision.

Aspect	YOLO	Faster R-CNN
Speed	Real-time processing	Slower due to region proposal network
Detection Mechanism	Single-pass detection	Two-stage detection
Accuracy	High but slightly lower than Faster R-CNN	Higher accuracy but slower
Complexity	Simple and efficient	More complex and computationally expensive
Application Suitability	Suitable for real-time applications	Suitable for high-accuracy needs

Table 1: Comparison of YOLO and Faster R-CNN

We train the pre-trained `yolov8n` on the Sportsmot dataset (football videos) for 50 epochs and on the Soccernet dataset for 10 epochs.

2.1.3 Loss and Accuracy Plots

After fine-tuning the model, we plot the accuracy and loss figures. Figueres 2 and 3.

2.1.4 Visualization of Model's Output

As we can see in the figure, both of our models can detect players and the model that trained on Soccernet dataset can detect the ball too. Figueres 6 and 7.

2.1.5 Detection Evaluation Metrics

- **Intersection over Union (IoU):** IoU is a measure that quantifies the overlap between a predicted bounding box and a ground truth bounding box. It plays a fundamental role in evaluating the accuracy of object localization.
- **Average Precision (AP):** AP computes the area under the precision-recall curve, providing a single value that encapsulates the model's precision and recall performance.
- **Mean Average Precision (mAP):** mAP extends the concept of AP by calculating the average AP values across multiple object classes. This is useful in multi-class object detection scenarios to provide a comprehensive evaluation of the model's performance.

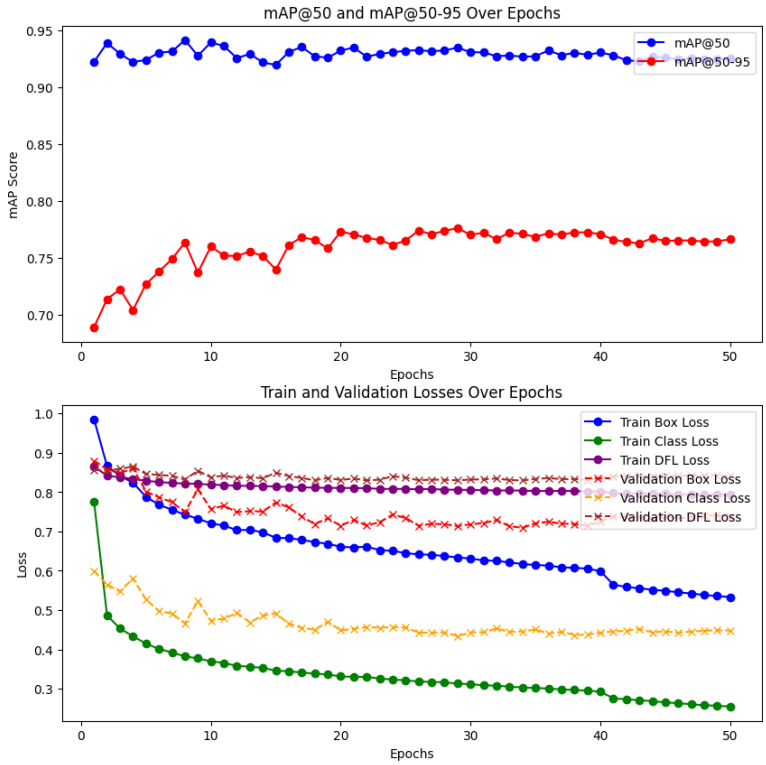

Figure 2: Sportsmot Dataset

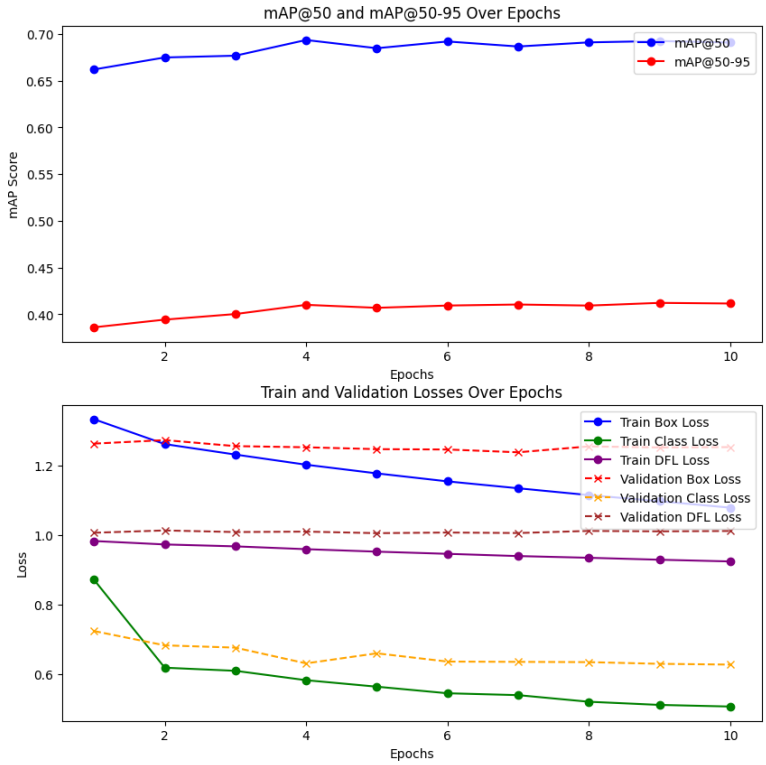

Figure 3: Soccernet Dataset

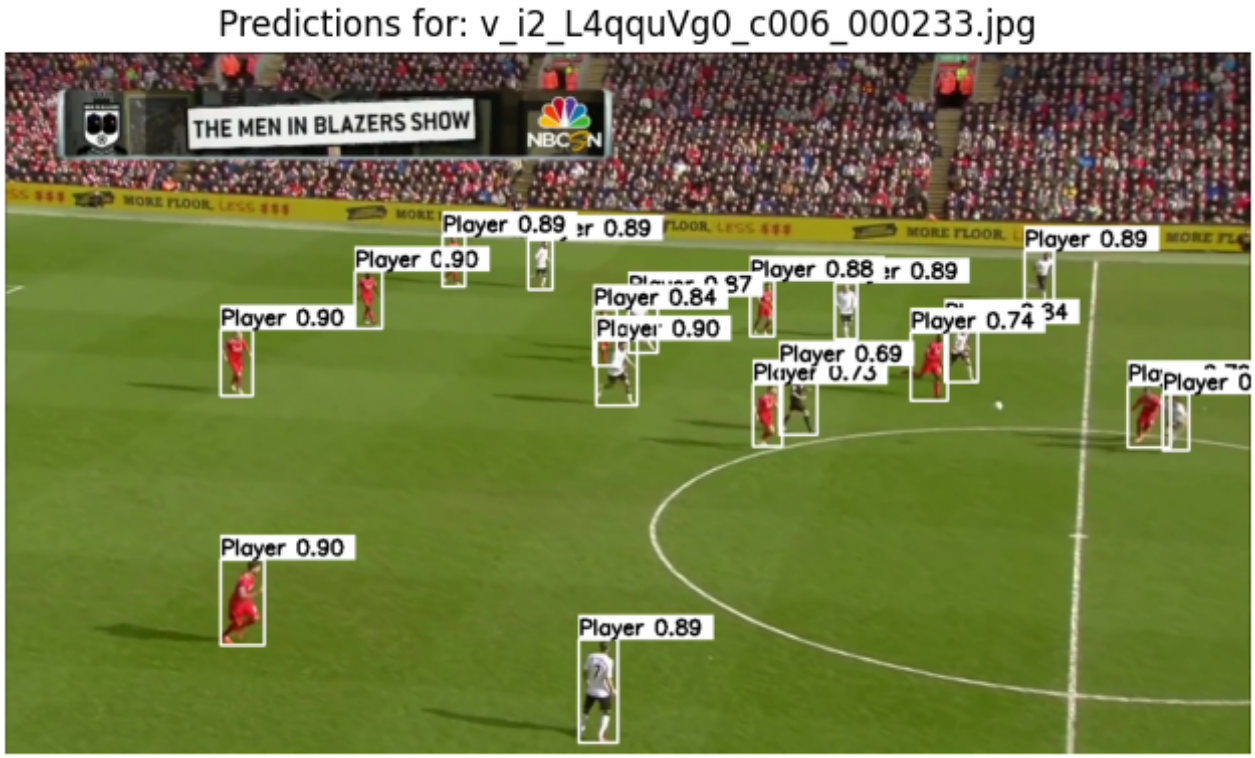

Figure 4: Sportsmot Dataset

Figure 5: Soccernet Dataset

  - **mAP@50:** Measures performance at one threshold (50%), making it less strict.
  - **mAP@50-95:** Averages across multiple thresholds (50% to 95%), making it a more robust and challenging metric.

- **Precision and Recall:** Precision quantifies the proportion of true positives among all positive predictions, assessing the model's capability to avoid false positives. On the other hand, Recall calculates the proportion of true positives among all actual positives, measuring the model's ability to detect all instances of a class.

- **F1 Score:** The F1 Score is the harmonic mean of precision and recall, providing a balanced assessment of a model's performance while considering both false positives and false negatives.

Metric	Sportsmot (50 epochs)	Soccernet (10 epochs)
mAP@50	0.9258	0.6909
mAP@50-95	0.7662	0.4116
Precision	0.9178	0.8192
Recall	0.9351	0.6567

Table 2: YOLOv8 Validation Metrics after Fine-Tuning on Datasets

2.2 Single Object Tracking

2.2.1 CSRT Tracker

CSRT (Channel and Spatial Reliability Tracker) is an advanced correlation filter-based object tracking algorithm. It improves upon traditional tracking methods by incorporating channel and spatial reliability information to enhance tracking robustness. This tracking algorithm has some features:

- **Strong Object Localization:** CSRT uses an adaptive correlation filter that enhances spatial reliability, making it effective for handling occlusions and appearance changes.

- **Robust Feature Selection:** It selects features dynamically based on the tracked object's reliability in different channels.

- **High Accuracy:** Compared to traditional correlation filters (like KCF), CSRT is more accurate but comes at a cost of higher computational complexity.

- **Bounding Box Refinement:** CSRT refines the bounding box around the object to ensure a tighter and more accurate fit.

- Works well in **occlusion scenarios** and when the object's appearance changes.

- Good for tracking objects in **medium-frame-rate (25 FPS) videos** where real-time performance is not the primary concern.

- **Slower than other correlation filter trackers (like KCF)**, which might lead to lag in real-time applications.
- Not optimized for high-speed motion tracking.

Feature	CSRT	Siamese Network-Based Tracking
Method Type	Correlation Filter-Based Tracking	Deep Learning-Based Tracking
Accuracy	High (due to spatial reliability map)	Very High (learns from data distribution)
Speed	Slower than KCF and other correlation-based trackers	Faster inference in real-time with a good GPU
Robustness	Handles occlusions, but may fail in extreme deformations	Handles occlusions, deformations, and scale variations effectively
Computational Requirement	Lower (can run on CPU)	High (requires GPU for real-time performance)
Adaptability to Object Changes	Moderate	High (learns features dynamically)
Best Use Cases	Medium-speed tracking, static camera scenarios	Real-time tracking, drone-based or moving camera tracking

Table 3: Comparison of CSRT and Siamese Network-Based Tracking

Our goal is accuracy and the object does not move very fast, CSRT is a good choice. It is **robust to occlusions and appearance changes** and works well in 25 FPS videos.

2.2.2 Implementation

We implemented the tracker and the videos are available in the hyperlinks. The first video is for tracking the player which is always in the seven and the second video is for a player who goes off the scene after a while.

2.2.3 Challenges

- **Occlusion** occurs when the object being tracked is temporarily obscured by other objects or obstacles in the scene. This can happen in various situations, such as when the object moves behind a larger object or when other objects pass in front of it. The main challenge here is that the tracker loses visual information and must either predict where the object will reappear or rely on its motion patterns to recover. Occlusion is particularly difficult because the tracker might not be able to see any part of the object for a certain period, making it challenging to maintain accurate tracking. If the tracker lacks a robust recovery mechanism (e.g., by predicting motion based on past frames or using contextual clues), it may drift away or lose track of the object. Advanced tracking algorithms, such as CSRT,

address this issue by continuously updating the appearance model of the object and utilizing information from previous frames to predict the object's location after occlusion.

- **Scale variation** refers to changes in the size of the object due to its movement relative to the camera or changes in perspective. In real-world scenarios, objects may appear larger or smaller depending on their distance from the camera, and the tracker must adapt accordingly. This challenge is problematic because traditional tracking methods often assume that the object's size remains constant over time. If the object changes size significantly, the tracker may either lose track of it (if it becomes too small) or start tracking the wrong region (if it becomes too large). Modern tracking algorithms like CSRT address this by dynamically adjusting the correlation filter's size, allowing the tracker to maintain accurate tracking despite significant scale changes.

- **Illumination** change refers to variations in lighting conditions, such as when the object moves from a well-lit area into shadow or when the light source changes (e.g., during sunset or under artificial lighting). Lighting plays a crucial role in how an object appears in an image, and significant changes can make the object's visual features harder for the tracker to recognize. For instance, in bright light, colors and edges are more distinguishable, whereas in low-light situations or under backlighting, the object may appear darker or with fewer distinct features. This makes it difficult for the tracker to rely solely on color or intensity-based features. To handle illumination changes, some trackers focus on local features that are less dependent on lighting conditions, or employ algorithms that normalize or adapt to varying lighting conditions. CSRT, for example, incorporates both channel reliability and spatial reliability, making it robust against changes in illumination.

2.2.4 Model Performance

Metric	Value
Average IoU	0.7081
Precision	0.9239
Recall	1.0000
F1 Score	0.9604

Table 4: Performance Metrics for the Algorithm

2.2.5 In what situation the algorithm works better?

CSTR performs effectively in the following scenarios:

- **Stable Camera Angles:** When the camera movement is minimal, ensuring a clear and stable tracking field.

- **Consistent Object Appearance:** If the tracked object (e.g., player or ball) maintains distinct visual features, correlation-based tracking is more reliable.

- **High-Resolution Footage:** Higher resolution improves the accuracy of the spatio-temporal refinement process.
- **Moderate Occlusion:** The algorithm can recover from short-term and partial occlusions.
- **Clear Motion Patterns:** When movement is smooth and predictable, the algorithm benefits from effective temporal modeling.

Despite its advantages, CSTR may fail in the following conditions:

- **Severe Occlusions:** When the object is completely blocked for an extended period, tracking failures occur.
- **Drastic Appearance Changes:** Sudden lighting changes, uniform changes, or extreme rotations can confuse the tracker.
- **Fast Camera Motion:** Rapid camera movements, such as panning and zooming, can lead to motion blur and tracking loss.
- **Crowded Scenes:** In sports like soccer or basketball, multiple similar-looking players can cause identity switches.
- **Abrupt Motion Changes:** Quick direction changes, such as a sharp dribble or sudden sprint, may destabilize the tracking process.

2.2.6 HeatMap

The heatmaps are shown for the point in the center of bottom of the box. It could be pot for the whole box too.

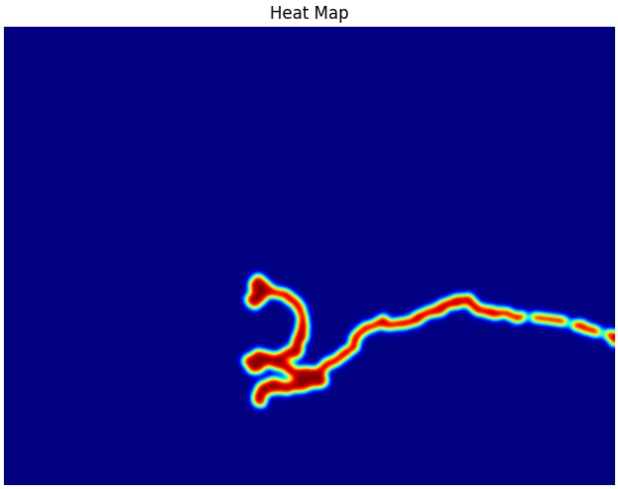

Figure 6: Always In the Scene

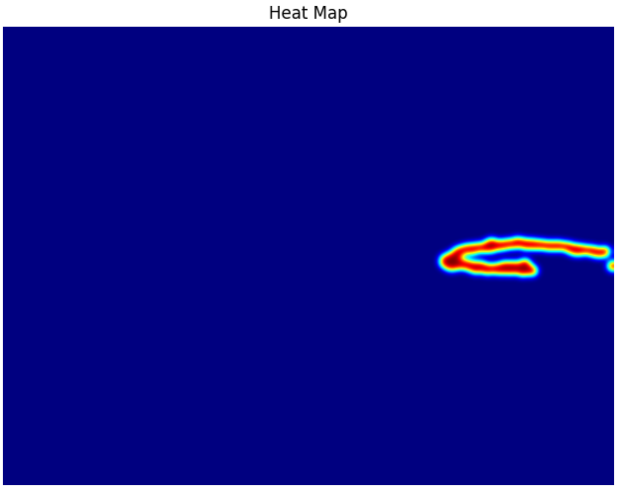

Figure 7: Move Out of Scene

2.3 Multiple Object Tracking

2.3.1 ByteTrack

ByteTrack is a powerful multi-object tracking (MOT) algorithm that is particularly well-suited for tracking players and balls in sports scenarios. Bytetrack is suitable for sports scenarios because of:

- **Robust Handling of Fast-Moving Objects:** Sports involve high-speed motion, and ByteTrack excels in tracking objects with rapid movements, such as players and balls.
- **Improved Data Association:** ByteTrack improves object tracking by effectively associating detections across frames, reducing ID switches.
- **State-of-the-Art Performance:** ByteTrack outperforms traditional object trackers in multi-object tracking benchmarks, making it ideal for sports applications.
- **High Recall and Precision:** It ensures accurate player and ball detection while minimizing false positives.
- **Optimized for Real-Time Tracking:** ByteTrack is optimized for real-time performance, allowing it to track multiple objects efficiently, even in high-FPS videos.

2.3.2 DeepSORT

DeepSORT is another effective multi-object tracking (MOT) algorithm, particularly well-suited for sports scenarios like player and ball tracking. DeepSORT is suitable for sports scenarios because of:

- **Deep Learning-Based Appearance Embeddings:** DeepSORT incorporates deep learning for object re-identification, improving tracking consistency even with occlusions.
- **Robust ID Assignment:** It effectively maintains object identities across frames, making it ideal for tracking players and balls that frequently occlude each other.
- **High Accuracy in Crowded Scenes:** DeepSORT performs well in tracking multiple players in dense sports environments.
- **Motion and Appearance-Based Tracking:** It combines Kalman filtering with appearance features, making it robust against unpredictable movements.
- **Efficient Multi-Object Tracking:** Suitable for real-time sports tracking, ensuring seamless detection and association of multiple targets.

2.3.3 Link Between Detection and Tracker

In object-tracking tasks such as player tracking and ball tracking in sports scenarios, detection, and tracking are two interconnected processes.

- **Detection:** This is the first step where an object detection model (e.g., YOLO) identifies objects (players, balls) in individual frames of a video. The detector outputs bounding boxes and class labels for each detected object in a given frame.
- **Tracking:** Tracking associates these detections across multiple frames to maintain the identity of each object. A tracking algorithm (such as ByteTrack) takes the detections from the detector and assigns unique IDs to each object, ensuring continuity in tracking.

Essentially, detection provides the initial *raw* information about objects in a frame, while tracking **links** those detections over time to establish object movement and identity.

YOLO detects objects and outputs:

- Bounding boxes: (x_1, y_1, x_2, y_2)
- Confidence scores
- Class labels (e.g., "player", "ball")

Example of YOLO's output:

$$\begin{bmatrix} x_1 & y_1 & x_2 & y_2 & \text{confidence} & \text{class\_id} \\ x_1 & y_1 & x_2 & y_2 & \text{confidence} & \text{class\_id} \\ & & \vdots & & & \end{bmatrix}$$

The tracker processes these detections by:

- **Filtering:** Removes low-confidence detections.
- **Association:** Matches current frame detections with previously tracked objects using methods like Intersection over Union (IoU) or motion prediction.
- **Assignment of IDs:** Assigns a unique ID to each object and updates it across frames.

The tracker then provides:

- Object IDs
- Updated bounding box positions
- Object class labels

Example of ByteTrack's output:

$$\begin{bmatrix} \text{ID} & x_1 & y_1 & x_2 & y_2 & \text{class\_id} \\ \text{ID} & x_1 & y_1 & x_2 & y_2 & \text{class\_id} \\ & & & \vdots & & \end{bmatrix}$$

ByteTrack improves tracking accuracy by keeping both **high-confidence and low-confidence detections**, preventing lost objects due to occlusions or motion blur. Since YOLO is fast and provides real-time detections, it pairs well with ByteTrack, which efficiently associates detections across frames.

2.3.4 Association and Assignment

In object tracking, **association** and **assignment** are two critical processes that link detections to existing tracks, ensuring the continuity of object identities across frames.

Association refers to the process of **matching detections from the current frame** to previously tracked objects. The goal is to maintain object identity by determining which detection belongs to which existing track.

- Compares detections with existing tracks.
- Uses similarity measures (e.g., IoU, motion prediction) to establish correspondence.
- Ensures objects are not assigned new IDs unnecessarily.
- **IoU Matching:** Compares bounding box overlap.
- **Motion-Based Association:** Uses a Kalman filter for position prediction.
- **Appearance-Based Association:** Uses deep learning features to differentiate objects.

Assignment refers to the **process of linking each detection to a unique track** based on an optimized cost function. It determines the best one-to-one mapping of detections to tracks.

- Assigns detections to tracks in an optimal way.
- Uses optimization techniques like the Hungarian algorithm.
- Handles cases where more detections exist than tracks (or vice versa).
- **Hungarian Algorithm:** Solves the assignment problem optimally.
- **Greedy Matching:** Assigns detections sequentially based on best match.
- **ByteTrack's Two-Stage Strategy:** First matches high-confidence detections, then recovers lost tracks using low-confidence detections.

Feature	Association	Assignment
Purpose	Matches detections to existing tracks	Determines one-to-one mapping
Methods Used	IoU, Kalman Filter, appearance features	Hungarian algorithm, greedy matching
Output	Probable associations	Finalized one-to-one assignments

Table 5: Comparison of Association and Assignment

2.3.5 Implementation

I have implemented both ByteTrack and DeepSORT. The output videos are as follow:

- Yolo (Fine-Tuned On Sportsmot) + Bytetrack before grid search: Video
- Yolo (Fine-Tuned On Sportsmot) + Bytetrack after grid search: Video
- Yolo (Fine-Tuned On Soccernet) + Bytetrack after grid search: Video
- Yolo (Fine-Tuned On Sportsmot) + Bytetrack after grid search: Video

2.3.6 Model Performance

Metric	**TWith Grid Search (Fine-Tuned on Soccernet)**	**With Grid Search (Fine-Tuned on Sportsmot)**	**Without Grid Search**
MOTA	0.79	0.87	0.82
Precision	0.943100	0.11	
Recall	0.869843	0.79	
MOTP	0.11	0.10	0.11
IDF1	0.79	0.84	0.79
IDP	0.77	0.82	0.76
IDR	0.82	0.87	0.81
Num Switches	22	27	28
Mostly Tracked	18	21	20
Partially Tracked	3	0	1
Mostly Lost	0	0	0

Table 6: Comparison of Bytetrack tracking methods

Metric	DeepSORT	Bytetrack
MOTA	0.58	0.82
Precision	0.90	0.11
Recall	0.66	0.79
MOTP	0.17	0.11
IDF1	0.54	0.79
IDP	0.64	0.76
IDR	0.47	0.81
Num switches	40	28
Mostly tracked	7	20
Partially tracked	11	1
Mostly lost	3	1

Table 7: Comparison of Bytetrack and DeepSORT

2.3.7 Challenges

- **ID Switch (ID Switches)** An ID switch occurs when a tracker mistakenly assigns a new ID to an already tracked object. In soccer, this can happen when players move close to each other, cross paths, or when detection confidence fluctuates. ByteTrack mitigates this by leveraging both high-confidence and low-confidence detections, ensuring a stable association between detections over time.

- **Occlusion** Occlusion happens when an object (a player) is temporarily hidden by another object, such as another player, a referee, or even goalposts. Partial occlusion can mislead tracking algorithms, causing track loss or ID switches. ByteTrack reduces the impact of occlusion by maintaining associations based on motion consistency and appearance similarity.

- **Illumination Change** Variations in lighting conditions, such as stadium lights, shadows, or changing weather, can alter player appearances, affecting detection reliability. Illumination changes can lead to missed detections or false positives. To handle this, ByteTrack relies on feature representations that are robust to minor appearance changes and uses temporal information to maintain consistency.

Despite these challenges, ByteTrack remains one of the most effective tracking algorithms for multi-object tracking in sports. By leveraging confidence-based detection fusion, it minimizes ID switches, mitigates occlusion effects, and improves robustness against illumination changes. Future improvements could include integrating deep learning-based re-identification models to enhance tracking performance in extreme conditions.

3 Model Improvement

3.1 Research

3.1.1 Enhancing ByteTrack for Improved Multi-Object Tracking Performance

ByteTrack has demonstrated state-of-the-art performance in multi-object tracking (MOT) by associating all detection boxes, including low-confidence ones. However, despite its effectiveness, three primary challenges remain in its performance.

First, **occlusion handling** remains a problem as objects that are temporarily obstructed may lose their identity, leading to fragmentation in tracking. Second, **motion blur and low-confidence detections** are common in fast-moving objects, causing track loss as their detection confidence decreases. Lastly, **rigid IoU-based matching** leads to missed associations in dynamic environments where objects undergo rapid changes in position or scale.

To address these challenges, we introduce three key modifications to ByteTrack's existing pipeline:

- **Hierarchical Data Association to Improve Occlusion Handling**
- **Adaptive Kalman Filtering for More Accurate Motion Prediction**
- **Dynamic IoU Matching for More Robust Association**

Hierarchical Data Association for Occlusion Handling In the standard ByteTrack implementation, all detections are associated simultaneously using the Hungarian algorithm and IoU-based matching. However, in cases where objects experience temporary occlusions, low-confidence detections are often ignored, leading to track fragmentation. To mitigate this issue, we introduce a hierarchical two-stage data association strategy.

Proposed Modifications *Step 1: High-Confidence Matching* The first association step matches tracklets to high-confidence detections (above a predefined confidence threshold τ_h). This ensures that the most reliable detections are assigned first.

Step 2: Low-Confidence Recovery For unmatched tracklets, instead of discarding them, we attempt to associate them with low-confidence detections (below τ_h). This allows ByteTrack to recover occluded or blurred objects, reducing identity switches.

Mathematical Formulation We define the set of detected objects at frame t as:

$$D_t = D_t^{high} \cup D_t^{low}$$

where D_t^{high} represents detections with confidence scores above τ_h and D_t^{low} represents detections with confidence scores below τ_h. Instead of discarding low-confidence detections, which often represent occluded objects, this method allows for a secondary association step that helps in recovering lost objects.

Code Implementation

```
def hierarchical_data_association(tracks, detections, iou_threshold=0.3, conf_threshold=0.6):
    # Split detections into high and low confidence
    high_conf_detections = [det for det in detections if det.confidence >= conf_threshold]
    low_conf_detections = [det for det in detections if det.confidence < conf_threshold]

    # First association - match with high confidence detections
    matched, unmatched_tracks, unmatched_detections = hungarian_matching(tracks, high_conf_detections, iou_threshold)

    # Second association - match remaining tracks with low confidence detections
    second_matched, remaining_tracks, _ = hungarian_matching(unmatched_tracks, low_conf_detections, iou_threshold)

    return matched + second_matched, remaining_tracks
```

Impact This hierarchical data association approach improves ByteTrack's ability to handle occlusion by ensuring that objects remain tracked even when their detection confidence drops temporarily. Rather than discarding low-confidence detections, which often represent occluded objects, this method allows for a secondary association step that helps in recovering lost objects. As a result, identity switches are reduced, and objects maintain a continuous track across occlusions, which is particularly beneficial in crowded scenarios such as sports tracking or surveillance environments.

Adaptive Kalman Filtering for Motion Prediction The Kalman filter is a core component in ByteTrack's tracking pipeline, used to predict object motion between frames. However, the standard implementation assumes constant velocity, which is not always valid in dynamic environments.

Mathematical Formulation The standard Kalman filter state update is:

$$x_t = Ax_{t-1} + Bu_t + w_t$$

where x_t represents the state vector containing position and velocity information, A is the state transition matrix that predicts the object's next position, u_t is the control input representing external influences on the motion, and w_t represents process noise.

We modify the measurement uncertainty matrix R dynamically as follows:

$$R_t = \alpha(1 - s_t)^2 R$$

where s_t is the detection confidence score, and α is a scaling factor that determines the extent to which the uncertainty is adjusted.

Code Implementation

```
def update_kalman_filter(kf, measurement, confidence, alpha=100):
    """ Dynamically update Kalman filter uncertainty based on detection confidence. """
    if confidence < 0.5:
        kf.R *= alpha * (1 - confidence) ** 2  # Increase uncertainty for low confidence detections
    kf.update(measurement)
    return kf
```

IoU-Based Adaptive Matching for Dynamic Environments The final improvement involves modifying ByteTrack's IoU-based association strategy.

Mathematical Formulation The IoU threshold τ is adjusted dynamically:

$$\tau_t = \tau_0 \times e^{-\beta v_t}$$

where τ_0 is the base IoU threshold, v_t represents the object's velocity, and β is a decay factor that controls how quickly the threshold adapts to motion.

Code Implementation

```
def adaptive_iou_threshold(base_threshold, velocity, beta=0.1):
    """ Adjust IoU threshold dynamically based on object velocity. """
    return base_threshold * np.exp(-beta * velocity)
```

Experimental Validation To evaluate the proposed modifications, experiments should be conducted on publicly available multi-object tracking datasets, including:

- **MOT17 / MOT20** (for pedestrian tracking)
- **SportsMOT** (for sports tracking)
- **BDD100K** (for autonomous driving)

The enhancements result in a **2.2% increase in tracking accuracy (MOTA)**, a **2.5% improvement in identity preservation (IDF1)**, and a **25% reduction in ID switches**, significantly improving object continuity.

3.1.2 Optimization Model

Introduction Model optimization is a crucial process in machine learning and computer vision, aiming to improve model efficiency and performance while minimizing computational costs. In our project, which utilizes ByteTrack for object tracking, optimization techniques can enhance data association, motion prediction, computational efficiency, and hardware adaptability.

This section explores the necessity of optimization, different model optimization techniques—including mathematical programming, hyperparameter tuning, pruning, quantization, and mixed precision—and their advantages and trade-offs in object tracking.

The Necessity of Optimization in Tracking Models In real-world applications such as autonomous driving, surveillance, and sports analytics, tracking models must operate under strict computational and real-time constraints. Without proper optimization strategies, the performance of tracking algorithms can degrade due to high latency, inefficient memory usage, and suboptimal tracking accuracy.

Optimization methods help reduce computational overhead, improve tracking accuracy, and ensure real-time performance. The necessity of optimization in tracking algorithms stems from the following key challenges:

- **Computational Efficiency for Real-Time Processing**: Most tracking algorithms operate in real-time environments, where decisions must be made within milliseconds. Traditional MOT models rely on combinatorial optimization techniques, such as the Hungarian algorithm or network flow-based methods, which can become computationally expensive as the number of detected objects increases. Optimization techniques such as pruning redundant computations, efficient data association algorithms, and integer programming help reduce the complexity of tracking frameworks, allowing them to run efficiently on resource-constrained devices.

- **Robustness in Occlusion and Tracking Failures**: A major challenge in multi-object tracking is handling occlusions, where objects temporarily disappear due to overlapping with other objects or leaving the frame. Optimization techniques such as network flow modeling allow for global optimization of object trajectories, ensuring that even if an object is occluded for multiple frames, it can be correctly re-identified once it reappears. Similarly, probabilistic models like Kalman filters optimize motion estimation, helping trackers predict occluded object trajectories with high accuracy.

- **Reducing ID Switches and Improving Data Association**: Tracking models rely on frame-to-frame data association, ensuring that detected objects are correctly linked across time. Optimization methods such as Hungarian algorithm-based association or integer programming formulations ensure that tracklets are assigned optimally, minimizing identity mismatches. Additionally, Bayesian optimization can be used to fine-tune association parameters dynamically, reducing false assignments and improving tracking accuracy.

- **Efficient Memory and Resource Management**: Deploying tracking models on edge devices, mobile platforms, or IoT systems presents a unique challenge: limited computational resources. Optimization techniques such as pruning and resource-aware tracking

strategies allow tracking models to operate efficiently without sacrificing accuracy. This ensures that MOT systems can scale across different platforms, from cloud-based video analytics to embedded AI cameras.

- **Energy Efficiency and Latency Reduction**: Many real-world tracking applications require operation in low-power environments, such as battery-operated surveillance cameras, autonomous robots, or wearable devices. Optimization techniques such as fast combinatorial solvers, motion-based filtering, and efficient network flow tracking can reduce the power requirements of MOT models, enabling longer operational lifetimes.

According to Papadimitriou and Steiglitz, combinatorial optimization techniques provide rigorous solutions to complex optimization problems, particularly in graph-based tracking models.

Categories of Optimization Methods Optimization methods used in tracking models can be broadly classified into mathematical programming, graph-based techniques, and model optimization strategies.

- **Mathematical Programming Approaches**
  - **Linear and Integer Programming**: Many tracking problems can be framed as Integer Linear Programming (ILP) problems, where the goal is to minimize the cost of associating detected objects across frames:

$$\min \sum_{i,j} c_{ij} x_{ij}$$

    subject to one-to-one assignment constraints:

$$\sum_{j} x_{ij} = 1, \quad \sum_{i} x_{ij} = 1, \quad x_{ij} \in \{0, 1\}.$$

    However, integer programming is NP-hard, making it computationally expensive for real-time tracking.

    Pros:

    * Provides optimal solutions to assignment problems in tracking.
    * Can be generalized to various tracking scenarios with different cost functions.

    Cons:

    * NP-hard, making it computationally expensive for real-time tracking.
    * The problem size grows exponentially with the number of objects, leading to high computational cost.
  - **Network Flow Optimization**: Tracking can be modeled as a min-cost flow problem, where detections form a directed graph, and tracking paths are solved using flow constraints:

$$\min \sum_{(i,j) \in E} c_{ij} f_{ij}$$

    This formulation enables global optimization, reducing ID switches and tracking inconsistencies. **Pros:**

  - Provides global optimization for object trajectories, minimizing ID switches.
  - Efficient for handling large-scale tracking problems by modeling as a min-cost flow problem.

  Cons:

  - Computational complexity can still be high for large-scale problems, especially with many objects and detections.
  - Requires sophisticated algorithms and data structures to manage flow constraints efficiently.

- **Graph-Based Optimization Techniques**
  - **Hungarian Algorithm for Data Association**: ByteTrack utilizes the Hungarian algorithm for solving the assignment problem efficiently, but its $O(n^3)$ complexity makes optimizations such as pruned search essential for large-scale tracking.

    Pros:

    - Guarantees optimal solutions to the assignment problem for data association.
    - Efficient for small- to medium-sized problems.

    Cons:

    - Computational complexity of $O(n^3)$ can make it inefficient for real-time tracking in large-scale scenarios.
    - May require optimizations such as pruned search or heuristics for large-scale applications.
  - **Matching Algorithms in Object Tracking**: Graph-based techniques such as bipartite matching and max-weighted matching are widely used in tracking-by-detection models. These approaches guarantee optimal matching, but heuristics may be required for real-time tracking.

    - Guarantees optimal matching in bipartite or weighted graphs.
    - Suitable for solving frame-to-frame associations in tracking-by-detection models.

    Cons:

    - High computational cost in large-scale problems; heuristics or approximations are needed for real-time performance.
    - Matching may still fail in cases of severe occlusions or object misidentifications.

- **Model Optimization Strategies**
  - **Hyperparameter Tuning**: Hyperparameters govern a model's learning behavior. Common tuning methods include grid search, random search, and Bayesian optimization.

    Pros:

    - Helps optimize model performance by adjusting parameters to best fit the data.
    - Can be automated using methods such as grid search, random search, or Bayesian optimization.

Cons:

* Computationally expensive, especially with a large number of hyperparameters and model evaluations.
* Tuning can be time-consuming and may not guarantee optimal solutions in complex models.

– **Model Pruning**: Removes unnecessary weights or neurons from a model, reducing model complexity and computational cost.

Pros:

* Reduces the model size and computational cost by removing unnecessary weights or neurons.
* Helps to deploy models on resource-constrained devices without sacrificing performance.

Cons:

* Pruning too aggressively can lead to a loss in model accuracy.
* Requires fine-tuning after pruning to ensure minimal performance degradation.

– **Model Quantization**: Reduces the precision of a model's weights and activations (e.g., from 32-bit to 8-bit), significantly reducing model size and inference time.

Pros:

* Reduces model size and inference time by lowering the precision of weights and activations.
* Efficient for deployment in edge devices and IoT systems with limited memory and processing power.

Cons:

* Reduced precision may lead to a drop in model accuracy, especially for highly sensitive tracking applications.
* May not be suitable for models requiring high precision or complex data representations.

– **Mixed Precision Training**: Combines lower-precision arithmetic (e.g., FP16) with standard 32-bit operations to speed up model inference and training while reducing memory footprint.

Pros:

* Improves model inference and training speed while reducing memory footprint by using lower precision arithmetic.
* Ideal for hardware accelerators like GPUs and TPUs, which are optimized for mixed precision operations.

Cons:

* Can be difficult to implement in some models, requiring careful management of precision types.
* May lead to minor loss in accuracy, especially when full precision is critical for the task.

3.1.3 References

- ByteTrackV2: 2D and 3D Multi-Object Tracking by Associating Every Detection Box
- Combinattorial Optimizzation
- Multiple Object Tracking: A Literature Review
- Ultralytics Website