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 Algorithms

1.1 SORT

1.1.1 Main Technique

The Simple Online and Realtime Tracking (SORT) algorithm is designed for efficient multi-object tracking. It primarily leverages a Kalman filter for state estimation and the Hungarian algorithm for data association to track objects across video frames.

1.1.2 Key Components

- **Kalman Filter:** For predicting object positions in the next frame.
- **Hungarian Algorithm:** For associating detections with existing tracks.
- **Bounding Box Detections:** Input from object detectors that indicate potential objects in each frame.

1.1.3 Algorithm Speed

SORT is recognized for its high processing speed, making it well-suited for real-time applications where quick decisions are critical.

1.1.4 Algorithm Performance

SORT performs reliably in environments with few occlusions and a limited number of objects. However, its performance may suffer in highly crowded scenes or in cases with significant object occlusion.

1.1.5 Complexity of Implementation

The implementation complexity of SORT is moderate. It strikes a balance between simplicity and performance by integrating well-understood algorithms (Kalman filter and Hungarian algorithm), making it accessible for rapid deployment.

1.1.6 Strengths

- **Real-Time Processing:** Optimized for speed.
- **Simplicity:** Easy to understand and implement.
- **Efficiency:** Effective in environments with low object density and minimal occlusion.

1.1.7 Weaknesses

- **Occlusion Handling:** Limited capability to manage prolonged occlusions.
- **Crowded Scenes:** Performance degradation when dealing with many overlapping objects.
- **Detection Dependency:** Highly reliant on the accuracy of the initial object detections.

1.1.8 When to Use SORT

SORT is ideal for real-time applications, such as surveillance or robotics, where objects are well-separated and computational resources are limited. It is best used when a balance between speed and tracking performance is required, and when the tracking environment is relatively simple.

1.1.9 Summary Table of SORT

Aspect	Description
Main Technique	Kalman Filter & Hungarian Algorithm
Key Components	Kalman Filter, Hungarian Algorithm, Bounding Box Detections
Algorithm Speed	High (Real-Time)
Algorithm Performance	Effective in low occlusion scenarios
Complexity of Implementation	Moderate
Strengths	Real-time, simple, efficient in sparse scenes
Weaknesses	Limited occlusion handling, struggles in crowded scenes
When to Use	Real-time tracking with minimal occlusions

Table 1: Summary of SORT Algorithm Characteristics

1.2 DeepSORT

1.2.1 Main Technique

DeepSORT extends the original SORT algorithm by integrating deep learning-based appearance descriptors. This allows the algorithm to better handle identity switches and improve tracking in crowded scenes. The core idea is to combine motion information (via a Kalman filter) with appearance features (obtained from a convolutional neural network) for robust data association.

1.2.2 Key Components

- **Kalman Filter:** Used for motion prediction of object trajectories.
- **Hungarian Algorithm:** Employed for associating predicted tracks with new detections.
- **Deep Appearance Descriptor:** A convolutional neural network extracts features that help in distinguishing objects.
- **Re-Identification (ReID) Module:** Uses appearance features to maintain object identities even during occlusions.

1.2.3 Algorithm Speed

DeepSORT generally runs at a high frame rate, though it is typically slower than SORT due to the additional computational overhead required for extracting deep features. The exact speed is dependent on the efficiency of the feature extractor and the available hardware.

1.2.4 Algorithm Performance

DeepSORT shows improved performance over SORT in scenarios with frequent occlusions and interactions between objects. The incorporation of appearance features significantly reduces identity switches, leading to more reliable tracking, especially in crowded scenes.

1.2.5 Complexity of Implementation

The implementation of DeepSORT is more complex than SORT. It requires:

- Integration of a deep neural network for feature extraction.
- Fine-tuning the ReID module to suit the tracking environment.
- Managing the additional computational resources and potential latency introduced by the deep learning components.

1.2.6 Strengths

- **Improved Identity Preservation:** Reduces identity switches by combining motion and appearance cues.
- **Robust to Occlusions:** Better handling of object occlusions and re-appearances.
- **Enhanced Performance in Crowded Scenes:** Utilizes deep features to distinguish between similar-looking objects.

1.2.7 Weaknesses

- **Increased Computational Overhead:** Requires more processing power due to deep feature extraction.
- **Implementation Complexity:** More challenging to implement and tune compared to SORT.
- **Real-Time Limitations:** May not be suitable for all real-time applications without sufficient hardware support.

1.2.8 When to Use DeepSORT

DeepSORT is best used in scenarios where:

- Tracking accuracy is critical, especially in environments with frequent occlusions.
- There is a need to maintain consistent object identities over long durations.
- The computational resources available can support the additional overhead of deep learning components.

Aspect	Description
Main Technique	Integration of Kalman Filter with deep appearance features
Key Components	Kalman Filter, Hungarian Algorithm, Deep Feature Extractor, ReID Module
Algorithm Speed	High frame rate, but slower than SORT due to extra computations
Algorithm Performance	Superior in handling occlusions and identity switches
Complexity of Implementation	Higher complexity due to deep learning integration
Strengths	Improved identity preservation, robust to occlusions, effective in crowded scenes
Weaknesses	Increased computational load, more complex implementation
When to Use	Scenarios requiring high tracking accuracy and reliable identity maintenance

Table 2: Summary of DeepSORT Algorithm Characteristics

1.3 FAIRMOT

1.3.1 Main Technique

FAIRMOT is a real-time multi-object tracking algorithm that unifies object detection and re-identification (ReID) into a single network architecture. This integrated approach ensures a balanced focus ("fairness") between the detection and ReID tasks, thereby improving tracking performance and reducing identity switches.

1.3.2 Key Components

- **Backbone Network:** Typically a network such as DLA (Deep Layer Aggregation) that extracts features from input images.
- **Detection Head:** Responsible for detecting objects and generating bounding boxes.
- **ReID Head:** Extracts discriminative appearance features used for object re-identification.
- **Feature Fusion Module:** Combines detection and ReID features to ensure balanced performance.

1.3.3 Algorithm Speed

FAIRMOT is designed for real-time applications. While its unified network design helps streamline processing, the inclusion of a ReID branch can introduce additional computational overhead compared to simpler trackers.

1.3.4 Algorithm Performance

FAIRMOT achieves robust tracking performance with low identity switch rates, particularly in crowded scenes. Its balanced design allows it to maintain accurate detections while effectively preserving object identities over time.

1.3.5 Complexity of Implementation

The implementation of FAIRMOT is moderately complex. Integrating detection and re-identification in a single network requires careful training and tuning. However, this unified approach simplifies the inference pipeline compared to multi-stage tracking systems.

1.3.6 Strengths

- **Balanced Performance:** Simultaneously optimizes detection and ReID, leading to fewer identity switches.
- **Real-Time Capability:** Suitable for real-time applications with competitive speed.
- **Robust in Crowded Scenes:** Handles dense environments effectively.

1.3.7 Weaknesses

- **Training Complexity:** Requires careful tuning of the joint detection and ReID tasks.
- **Computational Demand:** The integrated approach may demand more computational resources compared to simpler trackers.

1.3.8 When to Use FAIRMOT

FAIRMOT is ideal for scenarios where real-time tracking and accurate identity preservation are crucial, especially in crowded scenes or environments with complex interactions. It is well-suited for surveillance, autonomous driving, and other applications requiring robust multi-object tracking.

1.3.9 Summary Table of FAIRMOT

Aspect	Description
Main Technique	Unified network integrating detection and ReID for balanced performance
Key Components	Backbone (e.g., DLA), Detection Head, ReID Head, Feature Fusion Module
Algorithm Speed	Real-time capable, with additional overhead from the ReID branch
Algorithm Performance	Robust tracking with low identity switches, effective in crowded scenes
Complexity of Implementation	Moderate; requires joint training and careful tuning
Strengths	Balanced detection and ReID, real-time processing, robust in dense environments
Weaknesses	Higher training complexity and computational demand compared to simpler methods
When to Use	Applications needing reliable tracking and identity preservation in complex scenes

Table 3: Summary of FAIRMOT Algorithm Characteristics

1.4 Bytetrack

1.4.1 Main Technique

Bytetrack is a multi-object tracking algorithm that leverages high-quality detections along with both high-score and low-score detection candidates to effectively associate tracklets. It employs an IoU-based matching strategy to ensure robust and accurate tracking in real time.

1.4.2 Key Components

- **High-Quality Detector:** Provides robust detections with associated confidence scores.
- **IoU-based Association:** Uses Intersection over Union (IoU) to match detections with existing tracks.
- **Low-Score Detection Integration:** Incorporates lower-confidence detections to refine track associations.
- **Track Management:** Handles track initiation, continuation, and termination.

1.4.3 Algorithm Speed

Bytetrack is designed for real-time applications. Its efficient detection and association processes ensure low latency, making it suitable for high-speed tracking scenarios.

1.4.4 Algorithm Performance

Bytetrack demonstrates high tracking accuracy and robustness, especially in crowded scenes. Its strategy of using both high and low-score detections helps to reduce identity switches and maintain consistent tracking.

1.4.5 Complexity of Implementation

The implementation complexity of Bytetrack is moderate. While it builds upon existing object detection frameworks, integrating low-score detections and fine-tuning IoU thresholds requires careful calibration.

1.4.6 Strengths

- **High Accuracy:** Achieves robust tracking with minimal identity switches.
- **Real-Time Processing:** Suitable for applications that require fast and efficient tracking.
- **Robust Association:** Effective use of both high and low-score detections improves overall association quality.

1.4.7 Weaknesses

- **Dependency on Detector Quality:** The overall performance is heavily reliant on the quality of the underlying object detector.
- **Parameter Sensitivity:** Requires fine-tuning of detection thresholds and association parameters to achieve optimal results.

1.4.8 When to Use Bytetrack

Bytetrack is ideal for real-time tracking applications where high-quality detections are available. It is particularly well-suited for complex or crowded environments, such as surveillance or autonomous driving scenarios, where robust and accurate tracking is critical.

1.4.9 Summary Table of Bytetrack

Aspect	Description
Main Technique	Multi-object tracking using both high and low-score detections with IoU-based association
Key Components	High-quality detector, IoU-based matching, integration of low-score detections, track management
Algorithm Speed	Real-time capable with efficient detection and association processes
Algorithm Performance	High accuracy and robustness, especially in crowded scenes
Complexity of Implementation	Moderate; leverages existing detection frameworks with additional parameter tuning
Strengths	High accuracy, real-time processing, robust association
Weaknesses	Dependent on detector quality, requires fine-tuning of parameters
When to Use	Real-time tracking in environments with reliable detections and crowded scenes

Table 4: Summary of Bytetrack Algorithm Characteristics

1.5 Summary Table of Algorithms

Metric	SORT	DeepSORT	FairMOT	ByteTrack
Speed	60+ FPS	30-50 FPS	∼30 FPS	∼30 FPS
Performance	Moderate	Good	Very High	Very High
Complexity	Very Low	Moderate	Moderate	Moderate
Handles Occlusion	Poor	Good	Excellent	Excellent
Crowded Scenes	Weak	Good	Excellent	Excellent
Real-Time Usage	Excellent	Good	Excellent	Excellent
Resource Usage	Low	Moderate	Moderate	Moderate

Table 5: Summary Table of Algorithms

2 Algorithms Applications In Sport Scenarios

2.1 Review of related articles and analysis of the application of algorithms in sports scenarios

Many studies have explored the use of various computer vision and machine learning algorithms for tracking objects and players in sports environments. These algorithmic approaches have demonstrated significant potential in enhancing sports analytics and performance analysis.

One common technique is correlation filter-based tracking, which leverages efficient feature extraction and real-time processing capabilities to accurately follow the movements of individual players or balls. Research has shown that correlation filter algorithms can achieve precision rates exceeding 80% in dynamic sports settings, making them valuable for applications like player movement analysis and equipment tracking.

Siamese network architectures have also emerged as a powerful class of tracking algorithms in sports. These deep learning-based methods excel at robust feature representation and adaptive template matching, allowing them to maintain accurate tracking even in complex, occluded scenarios. Studies have highlighted the enhanced tracking accuracy and improved occlusion handling provided by Siamese network techniques.

Graph neural network (GNN) models represent another advanced algorithmic approach for sports tracking. GNNs excel at modeling the complex interactions and contextual relationships between multiple players or objects on the field of play. This holistic, graph-based understanding enables GNN-based trackers to tackle team sports scenarios with greater effectiveness, supporting applications such as tactical analysis and collective motion patterns.

More recently, transformer-based architectures have begun to revolutionize sports tracking, offering comprehensive contextual understanding, global feature representation, and attention-driven mechanisms. Researchers have demonstrated the superior performance of transformer-based trackers in handling challenging, long-range dependencies often present in dynamic sports environments.

Overall, the continued advancement of tracking algorithms has significantly expanded the capabilities of sports analytics, allowing for more granular player performance evaluation, detailed tactical insights, and enhanced biomechanical research. As computer vision and machine learning techniques evolve, we can expect to see even more sophisticated and versatile tracking solutions emerge to support the growing demands of the sports industry.

2.2 Single Object Tracking vs. Multiple Object Tracking

2.2.1 Overvoiew

Object tracking is a crucial aspect of sports analytics, used to monitor players, the ball, and referees in real-time. Two primary approaches to object tracking are:

- **Single Object Tracking (SOT)**: Focuses on tracking one specific object at a time.
- **Multiple Object Tracking (MOT)**: Tracks multiple objects simultaneously in a dynamic environment.

The choice between SOT and MOT depends on the specific use case, such as tracking a single player, a ball, or all entities on the field.

2.2.2 Comparison Between SOT and MOT

Table 6 shows this comparison.

2.2.3 Application in Sports Scenarios

- **Ball Tracking - Best Approach: SOT**

Feature	Single Object Tracking (SOT)	Multiple Object Tracking (MOT)
Definition	Tracks only one object at a time.	Tracks multiple objects simultaneously.
Computational Complexity	Lower; focuses on a single target.	Higher; involves detecting and updating multiple targets.
Use Case	Useful for tracking a specific player or ball.	Needed for tracking all players, ball, and referees.
Algorithms Used	Siamese Networks, KCF, CSRT	YOLO+DeepSORT, TrackNet, FairMOT
Challenges	Target loss due to occlusion or motion blur.	Data association, identity switching, occlusion handling.

Table 6: Comparison of Single Object Tracking (SOT) and Multiple Object Tracking (MOT)

  - The ball is a fast-moving object that requires precise tracking.
  - **Single Object Tracking (SOT)** is preferable as the focus is on one specific object.
  - **Algorithms:** Kalman Filter, CSRT, SiamMask, DeepSORT (for single object cases).

- **Player Tracking - Best Approach: MOT**
  - Multiple players need to be identified and tracked at once.
  - **MOT** is ideal for team sports like soccer, basketball, and football.
  - **Algorithms:** YOLO + DeepSORT, Bytrack, FAIRMOT.

- **Referee Tracking - Depends on Scenario**
  - **For a single referee:** SOT can be used if only one referee is being analyzed.
  - **For multiple referees in a match:** MOT is more efficient.
  - MOT is helpful when tracking their movement relative to players and the ball.

3 Tracking Metrics

In the context of multi-object tracking, various metrics are used to assess the performance and reliability of a tracker. Below are several key metrics along with explanations of what they measure and their significance.

3.1 metrics

3.1.1 Multi-Object Tracking Accuracy (MOTA)

What it Measures: MOTA is an aggregate metric that accounts for false positives (FP), false negatives (FN), and identity switches (IDSW). It is typically expressed as a percentage,

where a higher value indicates better overall tracking performance.

Details:

- **False Positives (FP):** Incorrect detections where an object is reported even though none exists.
- **False Negatives (FN):** Missed detections where an object present in the scene is not detected.
- **Identity Switches (IDSW):** Occurrences when a tracker changes the assigned ID of a target that should have a consistent identity.

3.1.2 Multi-Object Tracking Precision (MOTP)

What it Measures: MOTP quantifies the average overlap between the predicted bounding boxes and the ground truth bounding boxes (often using the Intersection over Union, IoU).

Details: A higher MOTP value indicates that the predicted positions are closer to the true object locations, reflecting better localization precision.

3.1.3 ID F1 Score (IDF1)

What it Measures: IDF1 is the F1 score (the harmonic mean of precision and recall) calculated specifically for object identities. It measures the tracker's ability to maintain consistent identities across frames.

Details:

- **Precision:** The fraction of correctly identified objects among all identified objects.
- **Recall:** The fraction of ground truth objects that were correctly identified.

3.1.4 Identity Switches (IDSW)

What it Measures: IDSW counts the number of times a tracked object's identity changes when it should remain consistent.

Details: Frequent identity switches indicate that the tracker struggles to maintain consistent object identities, which is critical in applications where re-identification is important.

3.1.5 Number of Switches

What it Measures: `num_switches` is a metric similar to IDSW. It records the total number of times an object's identity is switched during tracking.

Details: This metric provides a direct measure of the consistency of the tracking algorithm in preserving object identities. A lower number of switches implies a more stable tracker.

3.1.6 Mostly Tracked (MT) and Mostly Lost (ML)

What They Measure:

- **Mostly Tracked (MT):** The percentage of ground truth trajectories that are successfully tracked for more than 80% of their duration.
- **Mostly Lost (ML):** The percentage of ground truth trajectories that are tracked for less than 20% of their duration.

Details: A high MT value and a low ML value indicate that most objects are consistently tracked throughout the video.

3.1.7 Fragmentation (Frag)

What it Measures: Fragmentation counts the number of times a continuous ground truth trajectory is interrupted in the tracker's output.

Details: Frequent fragmentations imply that the tracker loses and then re-establishes a target's track repeatedly, suggesting instability in tracking continuity.

3.1.8 Recall and Precision

What They Measure:

- **Recall:** The fraction of ground truth objects that are correctly detected and tracked.
- **Precision:** The fraction of the tracker's detections that are correct (i.e., correspond to actual objects).

Details: A good tracking system should balance high recall (not missing objects) with high precision (minimizing false detections).

3.2 Limitations of Tracking Metrics and Potential Remedies

Tracking metrics such as MOTA, MOTP, IDF1, ID Switches, `num_switches`, MT/ML, Fragmentation, Recall, and Precision provide valuable insights into the performance of multi-object tracking systems. However, each metric has its limitations. In this document, we discuss the limitations of these metrics and suggest how combining or refining them can help mitigate their shortcomings.

3.2.1 Multi-Object Tracking Accuracy (MOTA)

Limitations:

- **Aggregation of Errors:** MOTA combines false positives, false negatives, and identity switches into a single value. This aggregation can mask the individual contributions of each error type.
- **Bias to Quantity:** A high MOTA score does not necessarily indicate good identity preservation or localization accuracy.

Potential Remedies:

- Use MOTA in conjunction with more specific metrics (e.g., IDF1 for identity preservation, MOTP for localization precision) to get a more nuanced evaluation.
- Report individual error components (FP, FN, IDSW) alongside MOTA.

3.2.2 Multi-Object Tracking Precision (MOTP)

Limitations:

- **Focus on Localization:** MOTP only measures the average overlap between predicted and true bounding boxes, ignoring the tracking continuity and identity consistency.
- **Insensitive to Detection Quality:** It does not account for missed detections or false positives.

Potential Remedies:

- Combine MOTP with recall and precision metrics to also capture detection performance.
- Use MOTP alongside metrics like IDF1 to balance localization accuracy with identity consistency.

3.2.3 ID F1 Score (IDF1)

Limitations:

- **Dependency on Correct Association:** IDF1 is sensitive to the matching strategy used for associating detections with ground truth.
- **Does Not Reflect Localization:** While it measures identity consistency, it does not account for the spatial accuracy of the detections.

Potential Remedies:

- Use IDF1 with MOTP to jointly evaluate identity consistency and localization accuracy.
- Refine the association method (e.g., thresholding and cost functions) to ensure a more reliable computation of IDF1.

3.2.4 Identity Switches (IDSW) and `num_switches`

Limitations:

- **Count-Based Metric:** Both metrics only count the number of switches without providing context about their duration or impact on overall tracking.
- **Sensitive to Short-term Errors:** A few transient switches can disproportionately affect the metric.

Potential Remedies:

- Complement these metrics with MT/ML (Mostly Tracked/Mostly Lost) to understand how long an object is tracked correctly.
- Consider weighted versions that penalize longer or more severe identity switches more heavily.

3.2.5 Mostly Tracked (MT) and Mostly Lost (ML)

Limitations:

- **Binary Thresholds:** The thresholds (e.g., 80% for MT and 20% for ML) are arbitrary and might not capture the full spectrum of tracking performance.
- **Ignoring Intermediate Cases:** Objects that are tracked between 20% and 80% of their lifespan are not specifically characterized.

Potential Remedies:

- Use continuous metrics or histograms of tracking durations to provide a more detailed view of tracking performance.
- Report the complete distribution of tracking durations instead of just binary thresholds.

3.2.6 Fragmentation (Frag)

Limitations:

- **Sensitivity to Short Interruptions:** Even minor interruptions in tracking (e.g., due to occlusion) can result in high fragmentation scores.
- **Does Not Quantify Impact:** Fragmentation counts do not indicate how severe the interruptions are with respect to tracking continuity.

Potential Remedies:

- Combine fragmentation with other continuity measures, such as the average duration of uninterrupted tracking segments.
- Refine the metric to discount very short interruptions that may be less impactful.

3.2.7 Recall and Precision

Limitations:

- **Isolation from Tracking Continuity:** While recall and precision measure detection performance, they do not capture the temporal aspect of tracking.
- **Ignores Identity Consistency:** These metrics do not account for identity switches or the consistency of tracking an object over time.

Potential Remedies:

- Use recall and precision in tandem with temporal metrics such as IDF1 or MOTA to evaluate both detection quality and tracking continuity.
- Incorporate tracking-specific modifications that account for the temporal association of detections.

4 Sport Tracking Dataset

4.1 Features of a Tracking Dataset

1. **High-Quality Annotations**
   - Bounding boxes for precise object detection.
   - Segmentation masks for detailed object boundaries.
   - Keypoints for pose estimation.
   - Unique object IDs for identity tracking.
   - Trajectory data for continuous tracking.
2. **Diverse and Realistic Scenarios**
   - Multiple camera angles for better generalization.
   - Different lighting conditions (indoor, outdoor).
   - Varied backgrounds and occlusions.
   - High-speed motion handling, especially for sports.
3. **Temporal Continuity**
   - Frame-by-frame tracking for smooth transitions.
   - Variable frame rates (30 FPS, 60 FPS, etc.).
   - Long video sequences for better tracking models.
4. **Multi-Object Tracking (MOT) Features**
   - Multiple objects in frame.

- Identity preservation across frames.
- Handling occlusions and reappearances.

5. **Standardized Benchmarking**
   - Predefined evaluation metrics (MOTA, MOTP, ID switches).
   - Publicly available training, validation, and test splits.
   - Baseline models for benchmarking.
6. **High-Resolution and High-Frame-Rate Videos**
   - HD or 4K footage.
   - Slow-motion support for high-speed analysis.
   - Consistent frame rates.

4.2 Datasets

1. **SportsMOT** A large-scale dataset specifically designed for multi-object tracking (MOT) in sports. Includes various sports like basketball, football, and volleyball. Provides high-resolution videos with annotated player bounding boxes.
2. **SoccerNet** A dataset designed for soccer (football) action recognition and event detection. Contains annotated events such as goals, shots, and passes. Includes tracking information for players and the ball.
3. **SoccerNet-V2 & SoccerNet-V3** Extended versions of SoccerNet, offering detailed tracking annotations, re-identification, and team/player tracking. Includes multiple camera angles and more events.
4. **NBA MOTS** A dataset designed for tracking basketball players. Contains multi-object tracking and segmentation annotations. Useful for applications like tactical analysis and game statistics.
5. **Tennis Track** A dataset for tennis player tracking and ball tracking. Captures high-speed ball movement and player positioning. Often used for player strategy analysis.
6. **Ice Hockey Tracking Dataset** A dataset focused on tracking players and pucks in ice hockey games. Includes player re-identification and action recognition.

5 Challenges in Object Tracking and Their Effects on Metrics

- **Occlusion:** When an object is partially or completely hidden behind another object, tracking becomes difficult. This increases *ID switches* and decreases *MOTA (Multiple Object Tracking Accuracy)*.
- **Illumination Change:** Variations in lighting conditions can cause tracking errors, leading to higher *false negatives* and lower *precision*.

- **Scale Variation:** Objects appearing at different scales due to camera zoom or perspective changes can reduce tracking consistency, affecting *MOTP (Multiple Object Tracking Precision)*.
- **Fast Motion:** Rapid object movement can result in motion blur and missed detections, decreasing both *recall* and *tracking accuracy*.
- **Similar Object Appearance:** When multiple objects look similar (e.g., players in the same uniform), Re-ID performance decreases, increasing *ID switches*.
- **Background Clutter:** Complex and dynamic backgrounds can cause false detections, lowering *precision* and increasing *false positives*.
- **Camera Viewpoint Changes:** Sudden viewpoint shifts may cause identity mismatches, reducing overall tracking stability and performance.

6 Dataloader Implementation (Sportsmot Dataset)

```
class SportsMOTDataset(Dataset):
    def __init__(self, root_dir, transform=None, augment=True,
    augmentation_prob=0.5):

        self.root_dir = root_dir
        self.transform = transform
        self.augment = augment
        self.augmentation_prob = augmentation_prob
        self.sequences = [d for d in os.listdir(root_dir) if os.path.isdir(
    os.path.join(root_dir, d))]
        self.samples = []
        self.seq_metadata = {}  # Store sequence-level augmentation choices

        for seq in self.sequences:
            seq_path = os.path.join(root_dir, seq)
            img_dir = os.path.join(seq_path, "img1")
            gt_file = os.path.join(seq_path, "gt", "gt.txt")

            try:
                annotations = pd.read_csv(
                    gt_file, header=None,
                    names=['frame', 'id', 'x', 'y', 'w', 'h', 'conf', 'cls',
     'vis']
                )
                annotations = annotations[annotations['cls'] == 1]  # Only
    include players
            except FileNotFoundError:
                print(f"Error: Annotation file not found for sequence: {seq}
    ")
                continue

            frame_files = sorted([
                f for f in os.listdir(img_dir)
                if f.endswith('.jpg') and f.split('.')[0].isdigit()
```

```
            ])

            # Decide augmentation for the whole sequence (Only Flip or Brightness)
            apply_augmentation = np.random.rand() < self.augmentation_prob  # Apply with given probability
            if apply_augmentation:
                aug_type = np.random.choice(["flip", "brightness"])
            else:
                aug_type = None  # No augmentation

            self.seq_metadata[seq] = {
                "augmentation": aug_type,
                "brightness_range": (5, 25) if aug_type == "brightness" else None,
                "max_brightness_factor": 1.5,
                "min_brightness_factor": 0.7
            }

            for frame_file in frame_files:
                frame_num = int(frame_file.split('.')[0])
                frame_anns = annotations[annotations['frame'] == frame_num]
                self.samples.append({
                    'image_path': os.path.join(img_dir, frame_file),
                    'annotations': frame_anns,
                    'sequence': seq,
                    'frame_num': frame_num
                })

    def __len__(self):
        return len(self.samples)

    import torchvision.transforms as transforms

    def __getitem__(self, idx):
        sample = self.samples[idx]
        image = cv2.imread(sample['image_path'])
        if image is None:
            raise ValueError(f"Failed to load image: {sample['image_path']}")
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)  # Convert BGR to RGB

        anns = sample['annotations']
        boxes = anns[['x', 'y', 'w', 'h']].values.astype(float)
        player_ids = anns['id'].values.astype(int)

        boxes[:, 2:] += boxes[:, :2]  # Convert (x, y, w, h)     (x1, y1, x2, y2)
        orig_height, orig_width, _ = image.shape

        #     Convert NumPy array to PIL Image using transforms.ToPILImage()
        to_pil = transforms.ToPILImage()
        image_pil = to_pil(image)

        if self.augment:
```

```
            seq_meta = self.seq_metadata[sample['sequence']]
            aug_type = seq_meta["augmentation"]

            if aug_type == "flip":
                image_pil = transforms.functional.hflip(image_pil)
                boxes[:, [0, 2]] = orig_width - boxes[:, [2, 0]]  # Adjust bounding boxes

            elif aug_type == "brightness":
                frame_num = sample['frame_num']
                brightness_range = seq_meta["brightness_range"]

                if brightness_range and brightness_range[0] <= frame_num <= brightness_range[1]:
                    midpoint = sum(brightness_range) / 2
                    if frame_num < midpoint:
                        factor = np.interp(frame_num, [brightness_range[0], midpoint],
                                           [1.0, seq_meta["max_brightness_factor"]])
                    else:
                        factor = np.interp(frame_num, [midpoint, brightness_range[1]],
                                           [seq_meta["max_brightness_factor"], seq_meta["min_brightness_factor"]])

                    image_pil = adjust_brightness(image_pil, factor)
                    image_pil = adjust_contrast(image_pil, factor)

        #     Apply final transformation pipeline (PIL Image     Tensor)
        if self.transform:
            transformed_image = self.transform(image_pil)  #     Now, this is a PIL Image

        else:
            transformed_image = transforms.ToTensor()(image_pil)  # Default tensor conversion

        boxes = torch.as_tensor(boxes, dtype=torch.float32)
        labels = torch.ones((len(anns),), dtype=torch.int64)

        return transformed_image, {
            'boxes': boxes,
            'labels': labels,
            'player_ids': torch.as_tensor(player_ids, dtype=torch.int64),
            'image_id': torch.tensor([idx]),
            'area': (boxes[:, 3] - boxes[:, 1]) * (boxes[:, 2] - boxes[:, 0]),
            'iscrowd': torch.zeros((len(anns),), dtype=torch.int64)
        }

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])
])

train_dataset = SportsMOTDataset(TRAIN_DIR, transform=train_transform,
    augment=True)
val_dataset = SportsMOTDataset(VAL_DIR, transform=train_transform, augment=
    False)

train_loader = DataLoader(
    train_dataset,
    batch_size = BATCH_SIZE,
    shuffle = True,
    num_workers = NUM_WORKERS,
    collate_fn = lambda x: tuple(zip(*x))
)

val_loader = DataLoader(
    val_dataset,
    batch_size = BATCH_SIZE,
    shuffle = False,
    num_workers = NUM_WORKERS,
    collate_fn = lambda x: tuple(zip(*x))
)
```

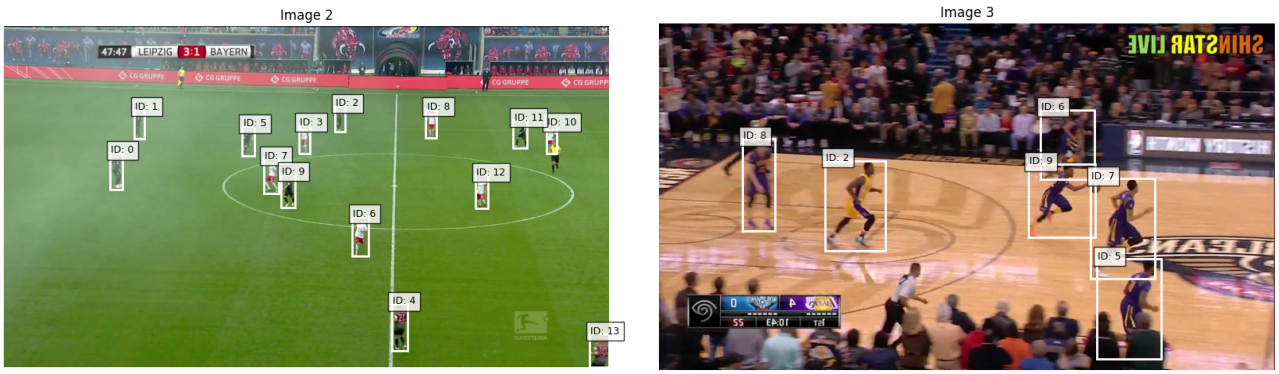

Figure 1: Two samples of Sportsmot dataset with their ground truth

7 Conclusion and Challenges

7.1 SORT

7.1.1 Challenges:

- extbfNo Re-Identification (Re-ID): Fails when objects are occluded and later reappear.
- extbfHigh ID-Switches: Due to simple IoU-based association.
- extbfStruggles with fast motion: Kalman Filter predictions are limited.

7.1.2 Suggestions to Improve:

- Integrate a Re-ID module to match objects after occlusion.
- Improve association with IoU + motion constraints.
- Use a better motion model, like a deep-learning-based state estimator.

7.2 DeepSORT

7.2.1 Challenges:

- extbfDependence on Appearance Features: Fails in similar-looking objects (e.g., players in sports).
- extbfHeavy Computation: Deep feature extraction slows real-time performance.
- extbfStruggles with Fast Occlusions: The Re-ID model needs improvements.

7.2.2 Suggestions to Improve:

- Use a hybrid association strategy (IoU + learned motion patterns).
- Optimize the feature extractor (use a lightweight CNN or transformer).
- Implement temporal smoothing in feature embedding to handle occlusions better.

7.3 FairMOT

7.3.1 Challenges:

- extbfTradeoff Between Detection and Embedding: Joint training of detection and Re-ID can degrade one of them.
- extbfSensitive to Occlusions: Despite Re-ID, occlusion handling is not perfect.
- extbfHigher Computation Cost than SORT/DeepSORT: Due to joint training.

7.3.2 Suggestions to Improve:

- Decouple detection and embedding training to avoid conflicts.
- Add an occlusion-aware feature enhancement to handle crowded scenes better.
- Use a teacher-student training setup to reduce computational overhead.

7.4 ByteTrack

7.4.1 Challenges:

- **High Computational Cost**: Uses both high-score and low-score detections for association.
- **Performance Drop in Low-Quality Detections**: Relies heavily on the quality of object detectors.
- **May Misassociate Re-appearing Objects**: Due to tracking all detections, some false positives may be included.

7.4.2 Suggestions to Improve:

- Use adaptive thresholding instead of a fixed confidence threshold.
- Implement a post-processing filter to eliminate wrongly associated low-confidence detections.
- Optimize by applying region-based tracking rather than global association.

7.5 Final Thoughts

- **SORT** $\rightarrow$ Needs Re-ID for long-term tracking.
- **DeepSORT** $\rightarrow$ Needs optimized feature extraction and better occlusion handling.
- **FairMOT** $\rightarrow$ Needs separate training for detection & tracking.
- **ByteTrack** $\rightarrow$ Needs efficient handling of low-score detections.

Tracker	Challenges	Suggestions
SORT	No Re-ID, High ID-Switches, Poor fast motion handling	Add Re-ID module, Improve IoU-based association, Use a better motion model
DeepSORT	High computation, Similar-looking object confusion, Occlusion issues	Optimize feature extraction, Hybrid IoU-motion association, Temporal smoothing
FairMOT	Detection-embedding trade-off, Occlusion sensitivity, High computation	Decouple detection and embedding training, Occlusion-aware feature enhancement, Teacher-student training setup
ByteTrack	High computation, Relies on detector quality, May misassociate objects	Adaptive thresholding, Post-processing filter for low-confidence detections, Region-based tracking

Table 7: Challenges and Suggested Improvements for Tracking Methods