Cricket Ball Detection & Tracking: Design Report

This project implements a cricket ball detection and tracking system using YOLOv8 object detection models combined with physics-aware interpolation and intelligent post-processing. The system is designed to handle the challenging task of tracking a small, fast-moving cricket ball in video sequences with high accuracy and robustness given a Cricket video from a single static camera.

Project Overview

Initial challenges identified:

Tracking a cricket ball in video presents several unique challenges:

1. **Small object size**: Cricket balls are small relative to frame resolution
2. **High velocity**: Balls move quickly, creating motion blur
3. **Occlusion**: Balls can be occluded by players, equipment, or camera angles
4. **Variable lighting**: Outdoor conditions create varying illumination
5. **Complex backgrounds**: Stadium environments with crowds, graphics overlays, and field markings

Approach

The solution employs a three-stage pipeline:

1. **Detection**: YOLOv8-based object detection for ball localization
2. **Tracking**: State machine with candidate selection and filtering
3. **Post-processing**: Physics-aware interpolation with outlier removal

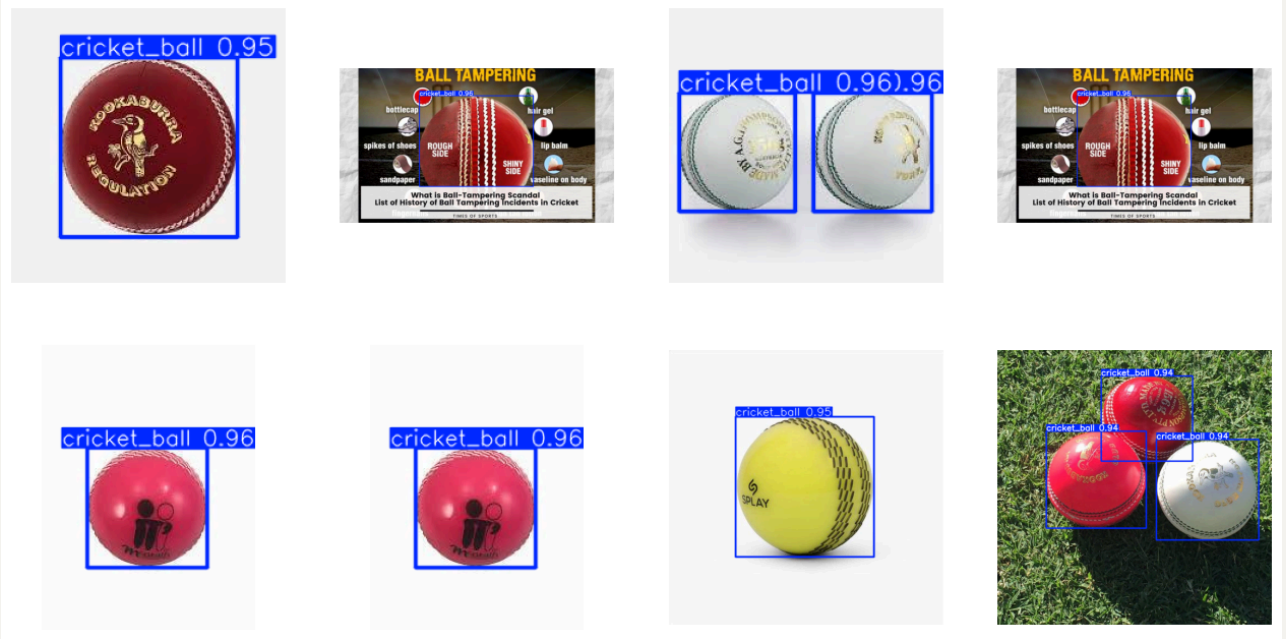
Dataset

The dataset used for training is the `cricket_ball_data` public dataset available in kaggle. To download :

```
import kagglehub

# Download latest version
path = kagglehub.dataset_download("kushagra3204/cricket-ball-dataset-for-yolo")

print("Path to dataset files:", path)
```



Modelling Choices

YOLOv8

For the core detection task, the **YOLOv8 (You Only Look Once)** architecture was selected.

Architecture Choice: YOLOv8 was chosen due to its state-of-the-art performance in real-time object detection, offering a critical balance between inference speed and spatial accuracy.

Training Configuration: The model was fine-tuned on a specific cricket ball dataset for 50 epochs with an image size of 640x640 pixels, ensuring the model learns the specific features of a cricket ball across various lighting and background conditions.

Model Variants Tested

The project experimented with multiple YOLOv8 variants:

MODEL	SIZE	USE CASE	REASON
YOLOv8s	Small	General detection	Balanced accuracy/speed, good for inference
YOLOv8m	Medium	Video tracking	Better accuracy for tracking pipeline
YOLOv12s	Small	Fine-tuned detection	Latest architecture with improved features

Key Finding: YOLOv8m (`cricket_ball_detector_model_v8m.pt`) performs better for tracking despite not being fine-tuned, likely due to:

- Better feature extraction for small objects
- More robust to motion blur
- Better generalization across video frames

yolov12s was restricted due to computation and time constraints

Training Configuration

Hyperparameters:

```
- Model: YOLOv8s (default) / YOLOv8m / YOLOv12s
- Epochs: 50 (default, configurable)
- Batch size: 8 (default, configurable)
- Image size: 640 (training), 1920 (inference)
- Freeze layers: 15 (transfer learning)
- Patience: 5 (early stopping)
```

Design Decisions:

TrackNET - Industry Standard

Research paper : https://arxiv.org/abs/1907.03698

This is a tailor made custom model for tracking high speed moving ball in sports applications and analysis such as tennis. However, the model is too complex to be implemented given my time and computation constraints. Additionally, didn't have access to the dataset use and focused on using the publicly available kaggle dataset.

YOLO

Thus, we used YOLO. It is simple, public and readily available while matching my computation power. YOLO datasets are publicly available with proper annotations.

1. **Transfer Learning with Layer Freezing**:
    - Freeze first 15 backbone layers to preserve pre-trained features
    - Only fine-tune detection head and final layers
    - **Reason**: Cricket balls share visual features with other spherical objects in COCO dataset
2. **Image Size Strategy**:
    - Training: 640×640 (standard YOLO size, faster training)
    - Inference: 1920×1920 (higher resolution for small ball detection)
    - **Reason**: Higher resolution at inference improves small object detection without slowing training
3. **Early Stopping**:
    - Patience of 5 epochs prevents overfitting
    - **Reason**: Small dataset size (1778 training images) requires

regularization

4. **Dataset Split**:
    - Train: 1778 images
    - Validation: 63 images
    - Test: 71 images
    - Standard 80/10/10 split ensures sufficient training data
5. **Confidence Threshold Strategy** : **Different thresholds for different tasks**:
    - **Inference (images)**: `conf=0.2` (higher threshold, fewer false positives)
    - **Tracking (video)**: `conf=0.005` (very low threshold, catch all detections)
    - Reason :
        - Video tracking can filter false positives through temporal consistency
        - Low threshold ensures no ball detections are missed
        - Post-processing removes noise

Training Analysis using Examples

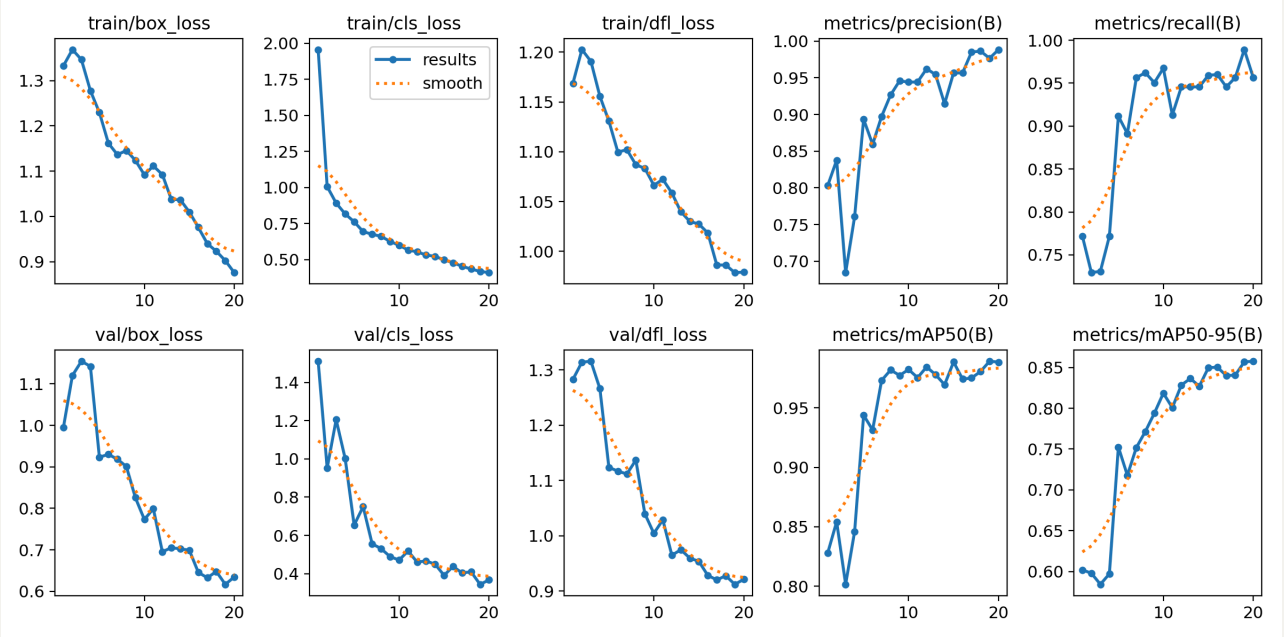
> *NOTE: The images shown are for 20 epochs training.*

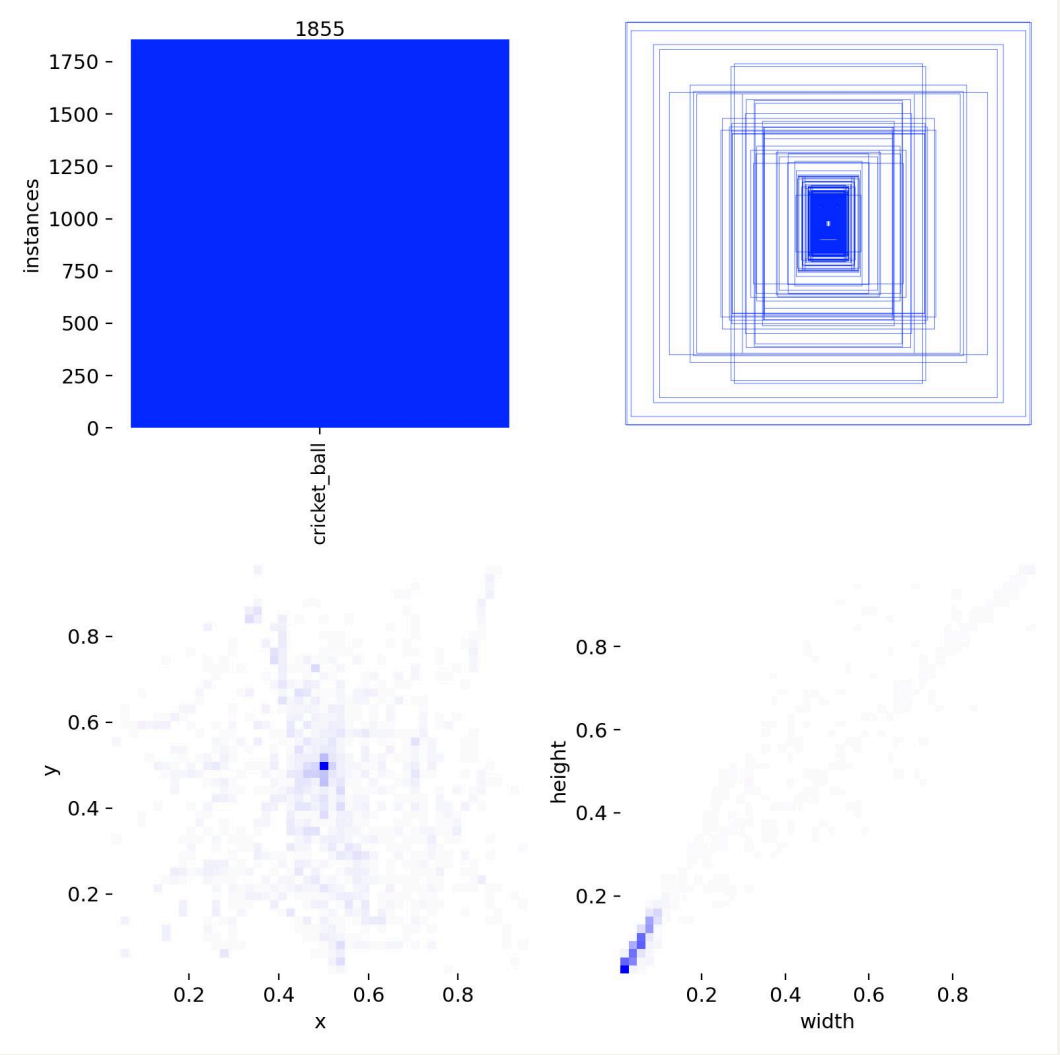

cricket_ball 0.7
cricket_ball 0.3
cricket_ball 1.0
cricket_ball 0.7
cricket_ball 0.8
cricket_ball 0.7
cricket_ball 0.5
cricket_ball 1.0
cricket_ball 0.4
cricket_ball 0.3
cricket_ball 0.8
cricket_ball 0.8
cricket_ball 0.9
cricket_ball 0.9
cricket_ball 0.4
cricket_ball 0.7
cricket_ball 0.8
cricket_ball 1.0

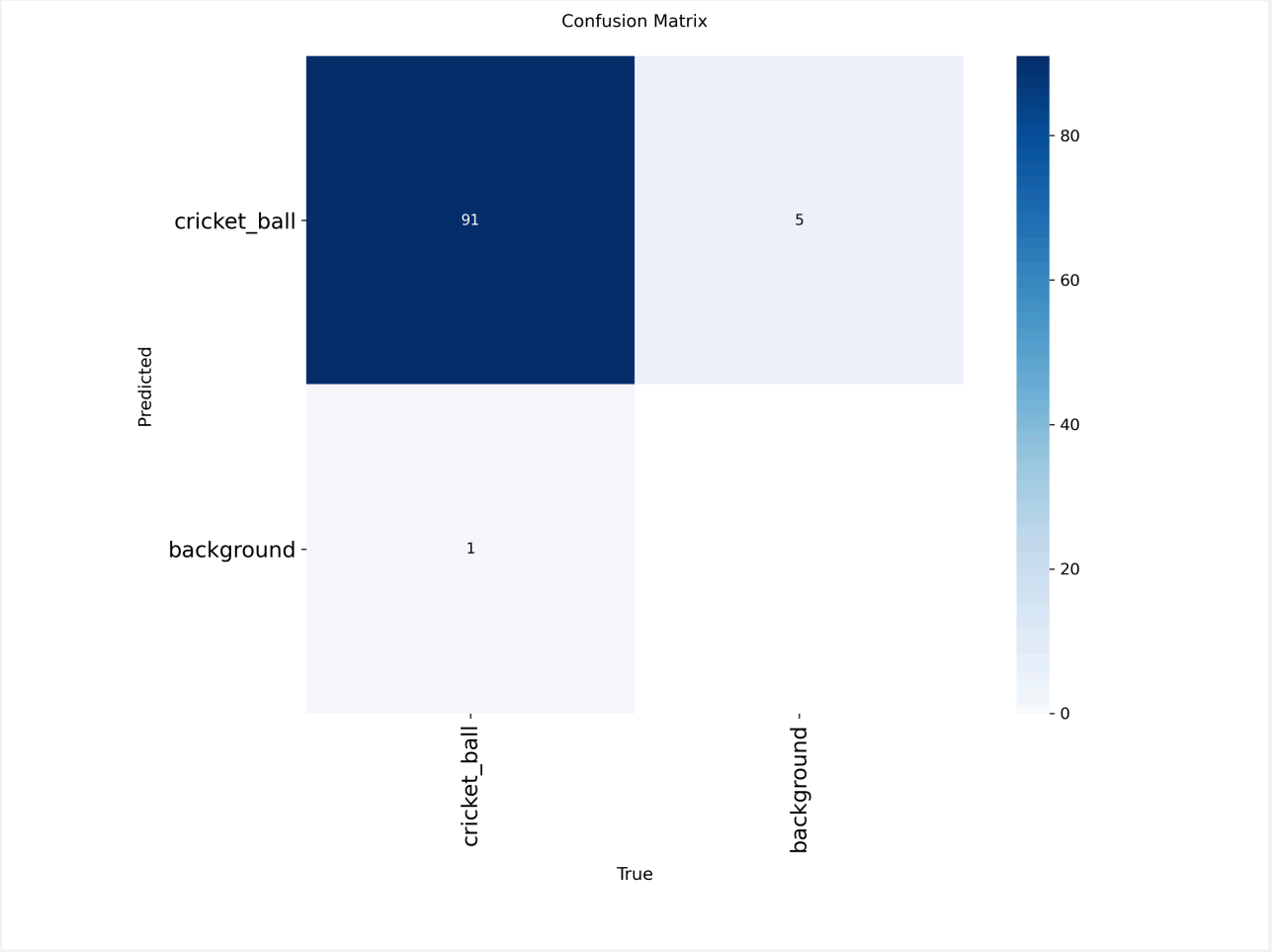

cricket_ball
cricket_ball
cricket_ball
cricket_ball
cricket_ball
cricket_ball
cricket_ball
cricket_ball
cricket_ball
cricket_ball
cricket_ball
cricket_ball
cricket_ball
cricket_ball
cricket_ball
cricket_ball
cricket_ball
cricket_ball

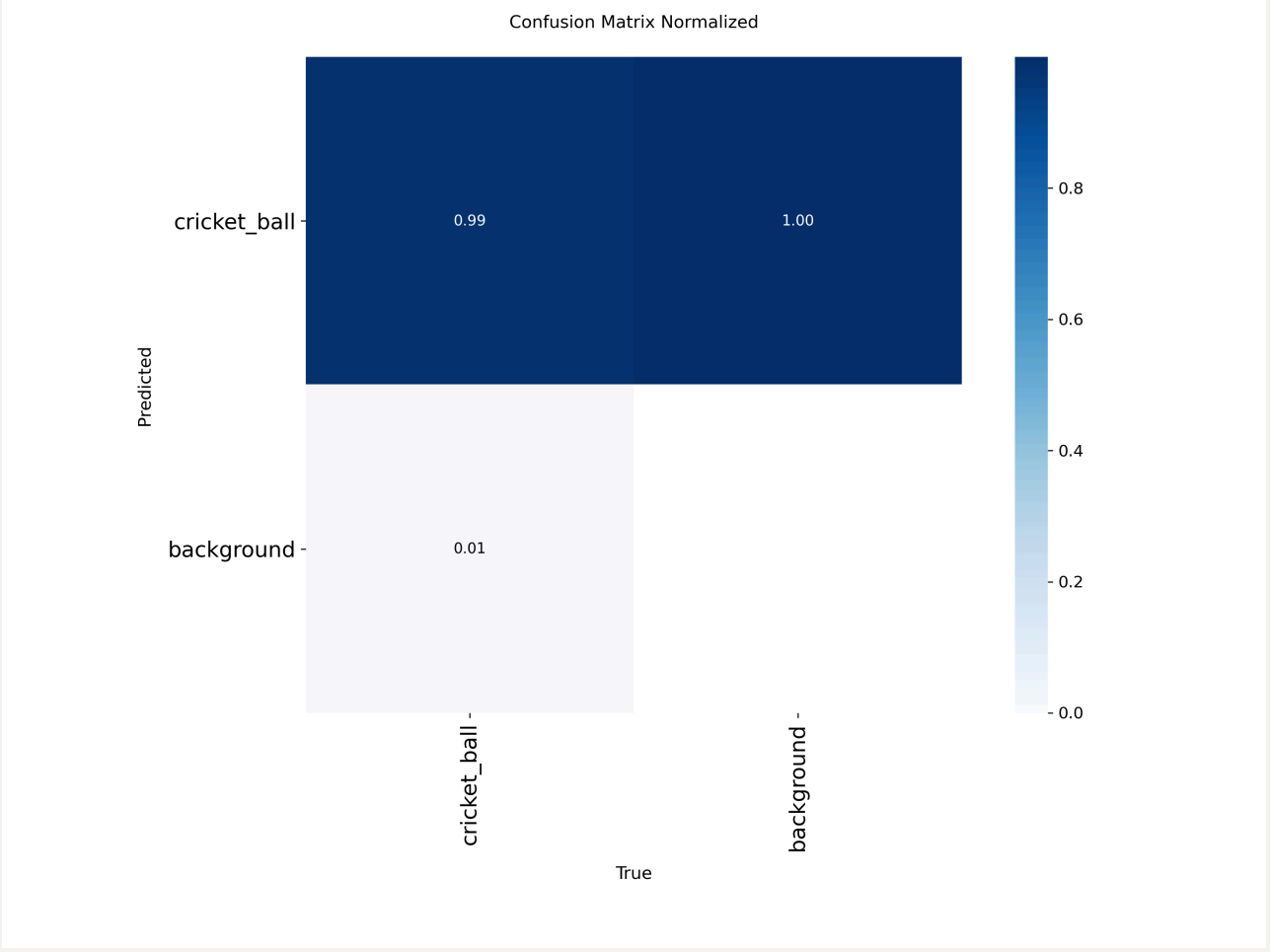

ball_img_1743.jpg
0
MOHAMMED SHAMI
INDIA
ball_img_5080.jpg
0
ball_img_2420.jpg
0
cricket_ball_314.jpg
0
ball_img_2794.jpg
0
ON BEHRENDORFF
ALIA
ball_img_5032.jpg
0
img_1253.jpg
0
ball_img_2939.jpg
0

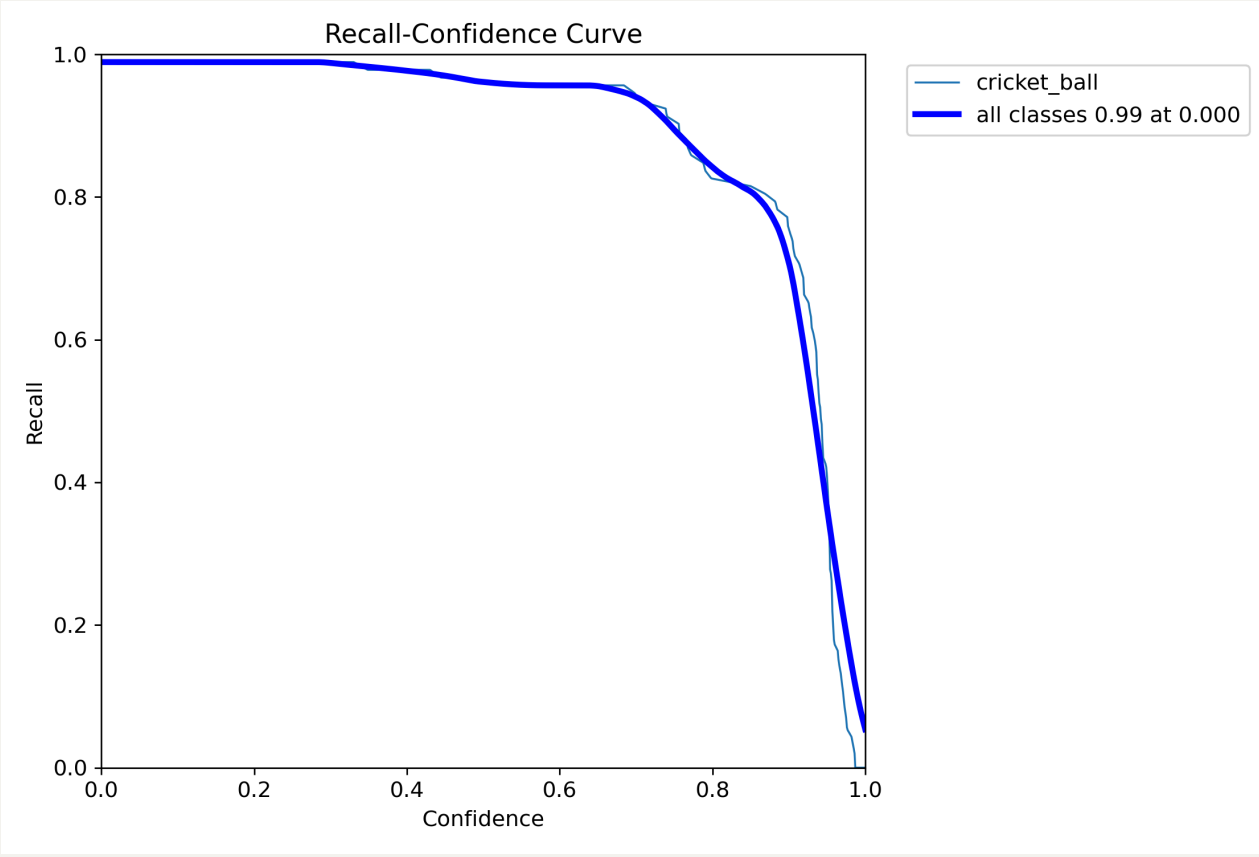

ball_img_4462.jpg
ball_img_839.jpg
ball_img_4461.jpg
ball_img_5056.jpg
ball_img_4431.jpg
ball_img_422.jpg
cricket_ball_59.jpg
ball_img_1720.jpg

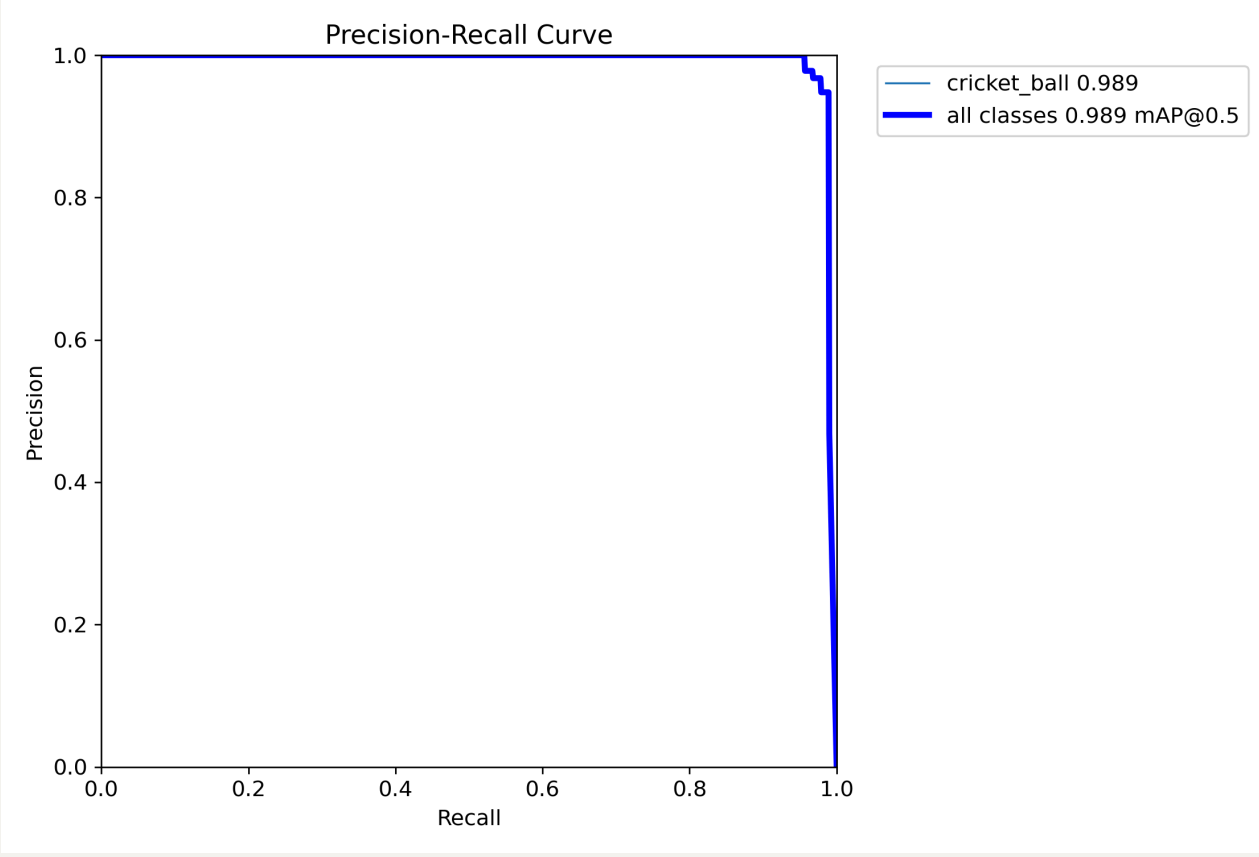

train/box_loss
train/cls_loss
results
smooth
train/dfl_loss
metrics/precision(B)
metrics/recall(B)
val/box_loss
val/cls_loss
val/dfl_loss
metrics/mAP50(B)
metrics/mAP50-95(B)

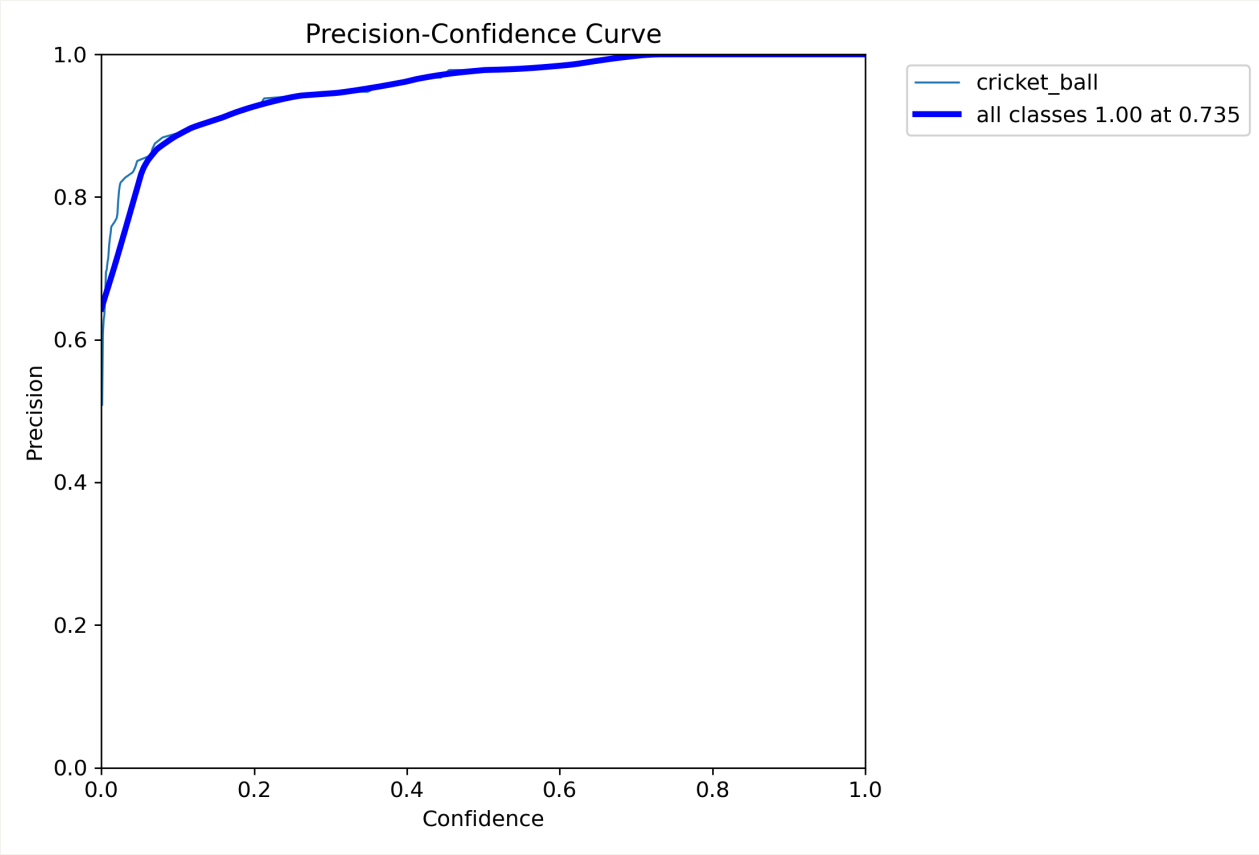

1855
1750
1500
1250
1000
750
500
250
0
instances
cricket_ball
0.8
0.6
0.4
0.2
y
0.2
0.4
0.6
0.8
x
height
width

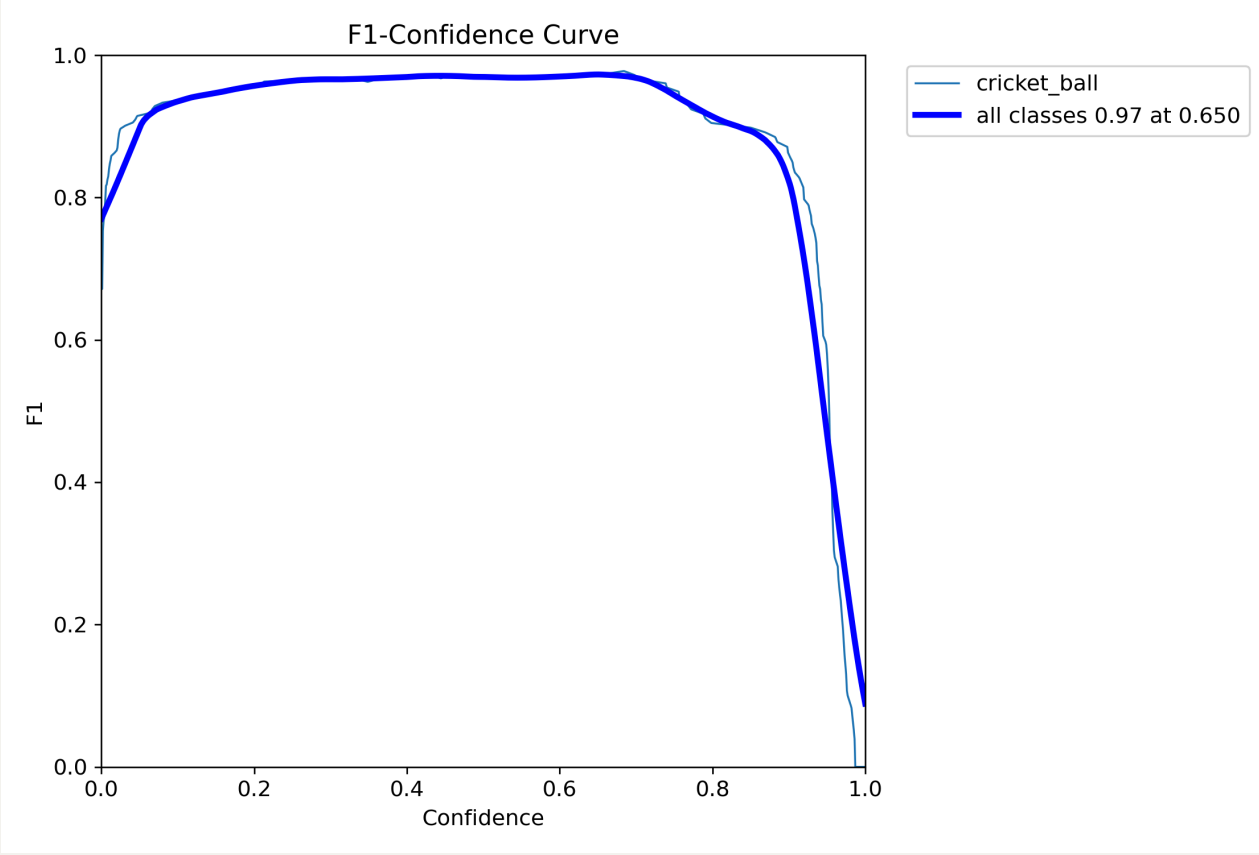

Confusion Matrix
cricket_ball
background
Predicted
91
5
1
cricket_ball
background
True
80
60
40
20
0


Confusion Matrix Normalized
cricket_ball
background
Predicted
0.99
1.00
0.01
cricket_ball
background
True
0.8
0.6
0.4
0.2
0.0


Recall-Confidence Curve
cricket_ball
all classes 0.99 at 0.000
Recall
Confidence
0.0
0.2
0.4
0.6
0.8
1.0


Precision-Recall Curve
cricket_ball 0.989
all classes 0.989 mAP@0.5
Precision
Recall
0.0
0.2
0.4
0.6
0.8
1.0


Precision-Confidence Curve
cricket_ball
all classes 1.00 at 0.735
Precision
Confidence
0.0
0.2
0.4
0.6
0.8
1.0


F1-Confidence Curve
cricket_ball
all classes 0.97 at 0.650
F1
Confidence
0.0
0.2
0.4
0.6
0.8
1.0

Tracking Pipeline Design

Two-Phase Architecture

The tracking system operates in two distinct phases:

Phase 1: Detection & State Machine

- Frame-by-frame detection using YOLO
- State machine with two modes: **Scanning** and **Tracking**
- Candidate selection based on confidence and distance

Phase 2: Post-Processing

- Streak noise removal
- Outlier detection and removal
- Physics-aware interpolation
- Gap filling

State Machine Design

States:

1. **Scanning Mode** (before lock-on):
    - Waiting for ball to appear in start zone
    - Validates detections against start zone ROI
    - Requires valid start position to transition
2. **Tracking Mode** (after lock-on):
    - Accepts detections based on distance/confidence
    - Tracks ball through frames

- Handles temporary occlusions

State Transition Logic:

```
if not has_locked_on:
    # Scanning: Only accept if in start zone
    if detection_in_start_zone:
        has_locked_on = True
        enter_tracking_mode()
else:
    # Tracking: Accept based on distance/confidence
    if candidate_within_dynamic_limit:
        accept_detection()
```

Reason:

- Prevents false positives from graphics/overlays before ball release
- Ensures tracking starts at correct moment
- Reduces computational overhead in early frames

Interpolation Strategy

Bounce Detection

Algorithm:

1. Find maximum Y-coordinate (lowest point in frame)
2. Split trajectory at bounce point
3. Interpolate pre-bounce and post-bounce segments separately

Reason:

- Ball trajectory changes at bounce (velocity reversal)
- Separate interpolation prevents smoothing across bounce discontinuity
- More accurate trajectory reconstruction

Post-Processing & Filtering

Multi-Stage Filtering Pipeline

The post-processing pipeline consists of four stages:

Stage 1: Streak Noise Removal

Function: `remove_streak_noise()`

Purpose: Remove isolated false positive detections before consistent tracking begins.

Algorithm:

- Scan from start of video
- Count consecutive detections (streak)
- Only keep detections after minimum streak length (default: 2 frames)

Reason:

- Prevents false positives from graphics/overlays in early frames
- Ensures tracking starts with consistent detections
- Reduces noise before main processing

Stage 2: Jump Outlier Removal

Function: `remove_jump_outliers()`

Purpose: Remove detections that jump unrealistically far between frames.

Algorithm:

- Calculate distance between consecutive detections
- Calculate gap size (frames between detections)

- Remove if `distance > max_jump_ratio × gap_size` (default: 100 pixels/frame)

Reason:

- Balls cannot move faster than physical limits
- Prevents tracking from jumping to distant false positives
- Maintains temporal consistency

Stage 3: Physics-Aware Interpolation

- Separate X/Y interpolation as described in Interpolation Strategy
- Fill gaps between valid detections
- Extrapolate beyond valid range if needed

Stage 4: Gap Cutting

Purpose: Remove interpolated segments across large gaps.

Algorithm:

- Identify gaps > `max_gap` frames (default: 15)
- Set interpolated values in large gaps to (-1, -1) (invalid)

Reason:

- Large gaps likely indicate ball is out of frame or occluded
- Interpolation across large gaps is unreliable
- Better to mark as missing than provide inaccurate positions

Detection Filtering

Function: `filter_detections()` in `utilities.py`

Filters Applied:

1. **Size filtering**:

- Minimum area: 40 pixels²
- Maximum area: 500 pixels²
- **Reason**: Cricket balls have predictable size range

2. **Aspect ratio filtering**:
    - Minimum: 0.7 (width/height)
    - Maximum: 1.6 (width/height)
    - **Reason**: Balls are approximately circular (aspect ratio ~1.0)

Reason: Removes detections that are clearly not balls (too large/small, wrong shape)

ROI Selection System

Interactive ROI Selection

Purpose: Allow users to define regions of interest to improve tracking accuracy.

Two ROI Types:

1. **Active Play Area**:
    - Defines where ball can appear
    - Everything outside is masked (set to black)
    - **Reason**: Excludes graphics overlays, scoreboards, crowd areas
2. **Start Zone**:
    - Defines where ball must first appear
    - Tracking only begins when ball detected in this zone
    - **Reason**: Prevents false positives before ball release

ROI Application

Active Area Masking:

```
def apply_active_area_mask(frame, roi_active):
    mask = np.zeros_like(frame)
    cv2.rectangle(mask, (ax, ay), (ax + aw, ay + ah), (255, 255,
255), -1)
    return cv2.bitwise_and(frame, mask)
```

Reason:

- Black masking prevents model from detecting objects outside play area
- Reduces false positives from graphics/overlays
- Improves tracking accuracy

Design Trade-offs

Accuracy vs. Speed

Decision: Prioritize accuracy for tracking, speed for training.

Reason:

- Tracking is offline/post-processing task (speed less critical)
- Training benefits from faster iteration
- Users can adjust image size based on needs

False Positives vs. False Negatives

Decision: Accept more false positives, filter in post-processing.

Reason:

- Low confidence threshold (0.005) catches all balls
- Post-processing removes false positives through temporal consistency
- Better to have false positives than miss ball detections

Manual ROI vs. Automatic

Decision: Provide interactive ROI selection with option to skip.

Reason:

- Manual ROI improves accuracy (excludes graphics)
- Optional for flexibility (users can skip if not needed)
- Default zones work for most cases

Physics Models: Simple vs. Complex

Decision: Use simple linear/quadratic models rather than complex physics simulation.

Reason:

- Simple models are sufficient for interpolation
- Complex models would require more parameters (air resistance, spin, etc.)
- Quadratic interpolation captures gravity effect adequately
- Faster computation

Limitations and Improvememts

Current Limitations

1. **Single Ball Tracking**: System tracks one ball at a time
2. **No Multi-Camera Support**: Single video input only
3. **Fixed Physics Models**: Doesn't account for air resistance, spin
4. **Manual ROI**: Requires user interaction (though optional)
5. **No Real-Time Processing**: Designed for offline video analysis

Potential Improvements

1. **Multi-Ball Tracking**:
    - Extend state machine to track multiple objects
    - Use object tracking algorithms (DeepSORT, ByteTrack)
2. **Advanced Physics Models**:
    - Air resistance modeling
    - Spin effects
    - More accurate trajectory prediction
3. **Automatic ROI Detection**:
    - Use semantic segmentation to detect field boundaries
    - Automatic start zone detection from player positions
4. **Real-Time Processing**:
    - Optimize for lower latency
    - Frame skipping strategies
    - Model quantization
5. **Multi-Camera Fusion**:
    - Combine detections from multiple camera angles
    - 3D trajectory reconstruction
6. **Deep Learning Tracking**:
    - Replace state machine with learned tracking model
    - End-to-end trainable tracking
7. **Data Augmentation**:
    - More aggressive augmentation during training
    - Synthetic data generation
8. **Model Ensembling**:
    - Combine multiple model predictions
    - Improve robustness
9. The dataset on which the object identification is done is a still image, when in video the ball moves at high speed appearing distorted due to motion blur. So a better training set which has moving images and videos would greatly improve the accuracy.

Analysis and Conclusions

Analysis on Test Videos

> *Test Video 8 :*

The trajectory is well define because the ball is moving at a relatively slower speed as it is a spin bowler, thus the motion blur distorts the ball less.

This works well for a lot of test videos. A few exceptions can be seen in video 1 and video 12, which can be reasoned asI:

1. In Video 1 there is too much motion blur based on our dataset which is still images, thus a random trajectory is detected. Additionally the video is very short, only about a second long. The bat itself merged with the ball, and it was difficult to see the position of ball with the naked eye.
2. The 12th video resembles a night scenario, while the model was trained mostly on well lit ball images and motion blur is high.

Conclusion

This cricket ball tracking system demonstrates a well-engineered approach to a challenging computer vision problem. The system successfully balances accuracy, robustness, and usability while maintaining a clean, modular codebase that can be extended for future improvements.