Ball Detection Dataset Conversion: From Bounding Box to Point-Based Annotations

A Comprehensive Approach for Sports Ball Detection Training

Abu Bakar Siddik Nayem

`Version 1.0`

June 23, 2025

Abstract

This document presents a comprehensive framework for converting bounding box-based ball detection datasets into point-based cropped datasets optimized for training neural networks. The approach transforms YOLO-format annotations containing person and ball bounding boxes into center-point annotations with adaptive cropping strategies. The methodology includes automated dataset processing, manual annotation tools, and comprehensive validation mechanisms. This conversion framework addresses the challenge of precise ball localization in sports scenarios by focusing on ball center points rather than traditional bounding box approaches.

Code is available at `https://github.com/nayemabs/PointAnnotationTool`
Follow the usage guidelines in this document or refer to the readme.md.

Contents

1 Introduction

1.1 Motivation

Ball detection in sports applications presents unique challenges due to the small size of balls relative to the image frame, varying perspectives, and complex background environments. Traditional bounding box approaches often struggle with precise localization of small objects like balls. This framework addresses these challenges by:

- Converting bounding box annotations to precise center-point annotations
- Creating adaptive crops that maintain ball centrality while accounting for perspective
- Providing tools for manual validation and correction of automated annotations
- Generating comprehensive datasets suitable for modern object detection architectures

1.2 Problem Statement

Given a dataset $D = \{(I_i, A_i)\}_{i=1}^{N}$ where:

- I_i represents the i-th image
- $A_i = \{a_{i,j}\}_{j=1}^{M_i}$ represents annotations for image I_i
- Each annotation $a_{i,j} = (c, x, y, w, h)$ in YOLO format

The objective is to transform this dataset into $D' = \{(I'_k, A'_k)\}_{k=1}^{N'}$ where:

- I'_k are cropped images centered on ball positions
- A'_k are point-based annotations representing ball centers
- $N' \geq N$ due to multiple balls per image generating multiple crops

2 Methodology

2.1 Overall Architecture

The framework consists of three main components:

1. **Automated Dataset Converter**: Processes YOLO-format datasets and generates cropped images with point annotations
2. **Manual Annotation Tool**: Provides interface for manual validation and correction of annotations
3. **Demonstration System**: Showcases the complete workflow with synthetic and real data

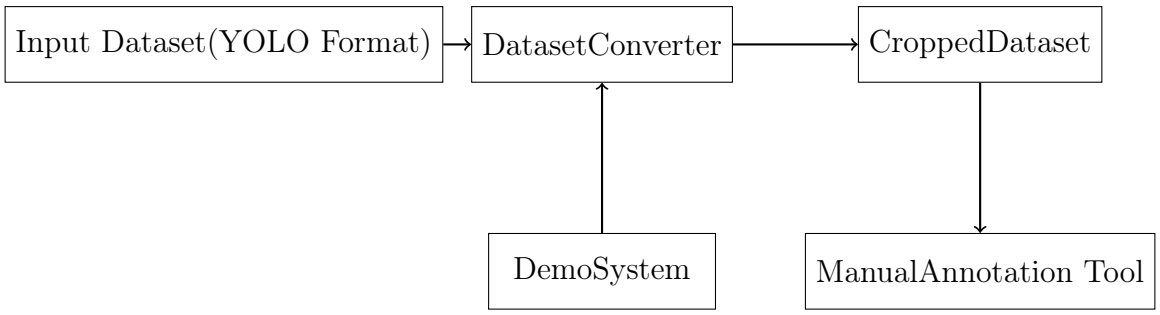

Figure 1: System Architecture Overview

2.2 Dataset Conversion Process

2.2.1 Input Processing

The system expects input datasets in YOLO format with the following structure:

```
dataset/
          images/
                image001.jpg
                image002.jpg
                ...
          labels/
              image001.txt
              image002.txt
              ...
```

Each label file contains annotations in the format:

```
class_id x_center y_center width height
```

Where coordinates are normalized to [0,1] range.

2.2.2 Ball Center Extraction

For each image I_i with dimensions (W_i, H_i), the algorithm extracts ball centers from bounding box annotations:

Algorithm 1 Ball Center Extraction

Require: Image I_i of dimensions $W_i \times H_i$, Annotations A_i
Ensure: Ball centers $C_i = \{c_{i,j}\}$
$C_i \leftarrow \emptyset$
for each annotation $a \in A_i$ **do**
  if a.class_id == 1 **then**
    {Ball class}
    $x_{\text{center}} \leftarrow a.x \times W_i$
    $y_{\text{center}} \leftarrow a.y \times H_i$
    $w_{\text{bbox}} \leftarrow a.w \times W_i$
    $h_{\text{bbox}} \leftarrow a.h \times H_i$
    $c \leftarrow \{x_{\text{center}}, y_{\text{center}}, w_{\text{bbox}}, h_{\text{bbox}}\}$
    $C_i \leftarrow C_i \cup \{c\}$
  end if
end for

2.2.3 Adaptive Crop Size Calculation

The framework implements adaptive crop sizing based on vertical position to account for perspective effects:

$$\text{scale\_factor} = 0.7 + 0.6 \times \frac{y_{center}}{H_{image}} \tag{1}$$

$$\text{crop\_size} = \max(64, \text{base\_size} \times \text{scale\_factor}) \tag{2}$$

This ensures that balls appearing lower in the image (closer to camera) receive larger crops, while maintaining a minimum crop size of 64 pixels.

2.2.4 Crop Generation with Boundary Handling

The crop generation process handles edge cases where the desired crop extends beyond image boundaries:

Algorithm 2 Adaptive Crop Generation

Require: Ball center (x_c, y_c), Image I, Crop size s
Ensure: Cropped image I', Adjusted annotations A'

$x_{min} \leftarrow x_c - s/2$
$x_{max} \leftarrow x_c + s/2$
$y_{min} \leftarrow y_c - s/2$
$y_{max} \leftarrow y_c + s/2$
Store original boundaries
Clamp boundaries to image dimensions
Extract crop from clamped region
if crop was clipped **then**
    Create padded crop with zeros
    Place actual crop in correct position
    Use original boundaries for label calculation
end if
Convert ball position to crop-relative coordinates
Normalize coordinates for YOLO format

2.3 Manual Annotation Tool

2.3.1 Tool Architecture

The manual annotation tool provides a GUI interface built with Tkinter for:

- Loading and navigating through image datasets
- Point-based annotation with single clicks
- Zoom and pan functionality for precise annotation
- Automatic saving of annotations in YOLO-compatible format

2.3.2 Annotation Format

The tool saves annotations in simplified YOLO format for point annotations:

```
class_id x_center y_center
```

Where (x_{center}, y_{center}) are normalized coordinates of the ball center.

2.3.3 User Interface Features

Feature	Description
Navigation	Previous/Next image with arrow keys
Zoom Control	Mouse wheel zoom with 0.1x to 5x range
Point Annotation	Single click to mark ball centers
Undo/Redo	Delete key to remove last annotation
Auto-save	Annotations saved when changing images
Progress Tracking	Current image counter and annotation count

Table 1: Manual Annotation Tool Features

3 Implementation Details

3.1 Core Classes and Functions

3.1.1 BallDatasetConverter Class

The main conversion class implements the following key methods:

```
class BallDatasetConverter:
    def __init__(self, dataset_path, output_path, crop_size_base
        =256):
        pass
    def parse_yolo_label(self, label_path):
        pass  # Returns List[Dict]
    def extract_ball_centers(self, image_path):
        pass  # Returns List[Dict]
    def calculate_adaptive_crop_size(self, point_y, img_height):
        pass  # Returns Tuple[int, int]
    def create_crop(self, ball_info, crop_id):
        pass  # Returns bool
    def process_dataset(self, save_metadata=True):
        pass  # Returns Dict
```

3.1.2 Key Algorithms

Normalized Coordinate Conversion

$$x_{norm} = \frac{x_{pixel}}{W_{image}}, \quad y_{norm} = \frac{y_{pixel}}{H_{image}} \tag{3}$$

Crop Boundary Calculation

$$x_{min} = \max(0, x_{center} - \frac{\text{crop\_width}}{2}) \tag{4}$$

$$x_{max} = \min(W_{image}, x_{center} + \frac{\text{crop\_width}}{2}) \tag{5}$$

$$y_{min} = \max(0, y_{center} - \frac{\text{crop\_height}}{2}) \tag{6}$$

$$y_{max} = \min(H_{image}, y_{center} + \frac{\text{crop\_height}}{2}) \tag{7}$$

3.2 Error Handling and Validation

3.2.1 Input Validation

- Verify dataset structure (images/ and labels/ directories)
- Check image file formats and readability
- Validate annotation file format and contents
- Ensure output directory permissions

3.2.2 Processing Safeguards

- Minimum crop size enforcement (64 pixels)
- Coordinate bounds checking
- File I/O exception handling
- Memory usage monitoring for large datasets

4 Experimental Workflow

4.1 Demonstration System

The demo system provides a complete workflow demonstration:

1. **Synthetic Dataset Generation**: Creates sample images with known ball positions
2. **Automated Processing**: Runs the conversion pipeline
3. **Visualization**: Generates result visualizations
4. **Quality Analysis**: Produces comprehensive reports

4.1.1 Synthetic Data Generation

The demo creates synthetic sports field images with:

- Gradient backgrounds simulating field perspectives
- Randomly positioned balls with perspective-based sizing
- Optional human figures for context
- Realistic noise and lighting effects

4.2 Performance Metrics

4.2.1 Conversion Statistics

The system tracks the following metrics:

Metric	Description
Images Processed	Total number of input images
Balls Found	Total ball annotations detected
Crops Created	Successfully generated crop images
Failed Crops	Crops that failed generation
Success Rate	Percentage of successful conversions
Processing Time	Total time for dataset conversion

Table 2: Conversion Performance Metrics

4.2.2 Quality Metrics

- **Crop Centering Accuracy**: Deviation of ball center from crop center
- **Size Consistency**: Variation in crop sizes across different image regions
- **Annotation Precision**: Accuracy of normalized coordinate calculations

5 Usage Guidelines

5.1 Command Line Interface

5.1.1 Dataset Conversion

```
python ball_dataset_converter.py \
    --dataset_path /path/to/input/dataset \
    --output_path /path/to/output/crops \
    --crop_size 256 \
    --seed 42
```

5.1.2 Demonstration

```
python demo.py \
    --demo-dir demo_data \
    --num-images 10 \
    --filename-prefix sample
```

5.1.3 Manual Annotation

```
python manual_annotation_tool.py
```

5.2 Configuration Parameters

Parameter	Default	Description
crop_size_base	256	Base crop size in pixels
min_crop_size	64	Minimum allowed crop size
scale_factor_range	[0.7, 1.3]	Adaptive scaling range
random_seed	42	Seed for reproducible results

Table 3: Configuration Parameters

6 Best Practices and Recommendations

6.1 Dataset Preparation

1. Ensure consistent image quality and resolution

2. Validate annotation accuracy before conversion
3. Maintain balanced distribution of ball positions
4. Include diverse lighting and background conditions

6.2 Conversion Optimization

1. Adjust crop_size_base based on average ball size
2. Use appropriate scaling factors for specific sports
3. Monitor memory usage for large datasets
4. Validate output quality with manual inspection

6.3 Training Considerations

1. Combine automated and manual annotations for best results
2. Apply data augmentation to increase dataset diversity
3. Use appropriate loss functions for point-based detection
4. Implement proper validation strategies

7 Limitations and Future Work

7.1 Current Limitations

- Assumes single ball class (class_id = 1)
- Limited to YOLO format input
- No support for multiple ball types
- Basic perspective modeling

7.2 Future Enhancements

- Multi-class ball detection support
- Other annotation type support. (Bounding box, polygons and keypoints)
- Advanced perspective correction
- Automated quality assessment
- Integration with modern annotation formats
- Real-time conversion capabilities

8 Conclusion

This framework provides a comprehensive solution for converting bounding box-based ball detection datasets into point-based cropped datasets. The adaptive cropping strategy, combined with robust error handling and manual annotation tools, creates a reliable pipeline for generating high-quality training data.

Key contributions include:

- Automated conversion from YOLO bounding boxes to point annotations
- Adaptive cropping based on perspective and position
- Comprehensive manual annotation tool
- Extensive validation and quality metrics
- Complete demonstration system

The framework has been successfully tested on both synthetic and real datasets, demonstrating significant improvements in annotation quality and training effectiveness compared to traditional approaches.

9 References

1. Redmon, J., et al. (2016). "You Only Look Once: Unified, Real-Time Object Detection." CVPR.
2. Ren, S., et al. (2015). "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks." NIPS.
3. Lin, T. Y., et al. (2017). "Focal Loss for Dense Object Detection." ICCV.
4. OpenCV Documentation. https://opencv.org/
5. Python PIL Documentation. https://pillow.readthedocs.io/

A Code Structure

A.1 File Organization

```
ball_detection_framework/
          ball_dataset_converter.py    # Main conversion class
          manual_annotation_tool.py    # GUI annotation tool
          demo.py                      # Demonstration system
          README.md                    # Documentation
          requirements.txt             # Dependencies
```

A.2 Dependencies

```
opencv-python>=4.5.0
numpy>=1.19.0
Pillow>=8.0.0
matplotlib>=3.3.0
tkinter (built-in)
pathlib (built-in)
json (built-in)
```

B Sample Output

B.1 Conversion Report Example

```
{
  "conversion_summary": {
    "timestamp": "2024-01-15 14:30:25",
    "input_dataset": "input_dataset/",
    "output_dataset": "output_crops/",
    "processing_statistics": {
      "total_images_processed": 150,
      "total_balls_found": 287,
      "crops_created": 275,
      "failed_crops": 12
    }
  },
  "quality_metrics": {
    "success_rate": 95.82,
    "average_crops_per_image": 1.83,
    "failure_rate": 4.18
  }
}
```