Report Edgefleet project:

Introduction

Object detection is an essential task in computer vision used for identifying and localizing objects in images and videos.

In this project, a custom YOLO11-based detection system was developed using a dataset generated from video frames. The goal was to create a robust model capable of detecting the target object in real-world video sequences.

This report describes dataset creation, annotation, preprocessing, model training, evaluation, and results.

Dataset Preparation

Frame Extraction

A total of **373 frames** were extracted from a recorded video. These frames captured different angles, lighting conditions, and object positions to ensure dataset diversity.

Annotation Process

All 373 frames were uploaded to **Roboflow**.

Objects were manually annotated using bounding boxes.
Roboflow processed annotations and exported the dataset in **YOLO format**.

DIFFERENT DATASET PREPARATION

Before creating my own dataset, I tried using some existing cricket-ball datasets and ready-made models. But none of them worked properly for my video.

1. Kaggle Cricket Ball Dataset

I first tried a cricket ball dataset from Kaggle that was already annotated.
But it didn't work because:

- The camera angle was totally different,
- The ball looked different in size and speed,
- The background and lighting did not match my video at all.

So the model got confused and could not detect the ball correctly in my video.

2. TrackNet Pretrained Weights

Next, I tried TrackNet, a model specially made for cricket ball tracking.
I used the pretrained weights, hoping they would work out of the box.

But they also failed because:

- The pretrained model did not match my video style,
- It missed the ball many times,
- Sometimes it gave completely wrong results.

TrackNet was trained on TV broadcast-style cricket videos, while my video was very different.
So it couldn't understand my frames properly.

3. Manual Annotation (Final Working Solution)

After these failures, I finally decided to **manually annotate the frames** from my own video using Roboflow.

This worked the best because:

- The annotations matched my exact camera angle,
- Only useful frames were included,
- The model learned how the ball looks in *my* environment,
- There was no confusion between training and testing data.

As a result, the YOLO11 model trained on my manually annotated dataset gave **much better detection** and **more stable results** than all the previous attempts.

Dataset Folder Structure (Explained in Words)

The dataset is organized into two main folders: **images** and **labels**.

1. **images/**
    a. This folder contains all the image files.
    b. It is further divided into three parts:
        i. **train/** → images used for training the model
        ii. **val/** → images used for validation
        iii. **test/** → images used for final testing after training
2. **labels/**
    a. corresponding label files for each image.
    b. Each label file the cricket ball class, and bounding box coordinates.
    c. It also has the same three folders:
        i. **train** -labels for training images
        ii. **Val -**labels for validation images
        iii. **Test -l**abels for test images

3. **data.yaml.**

```
train: dataset/images/train
val: dataset/images/val
test: dataset/images/test

nc: 1
names: ["cricket ball"]
```

Model Training.

I selected **YOLO11** from Ultralytics for this project because.

It gives high accuracy

It works in real-time

It is simple to train

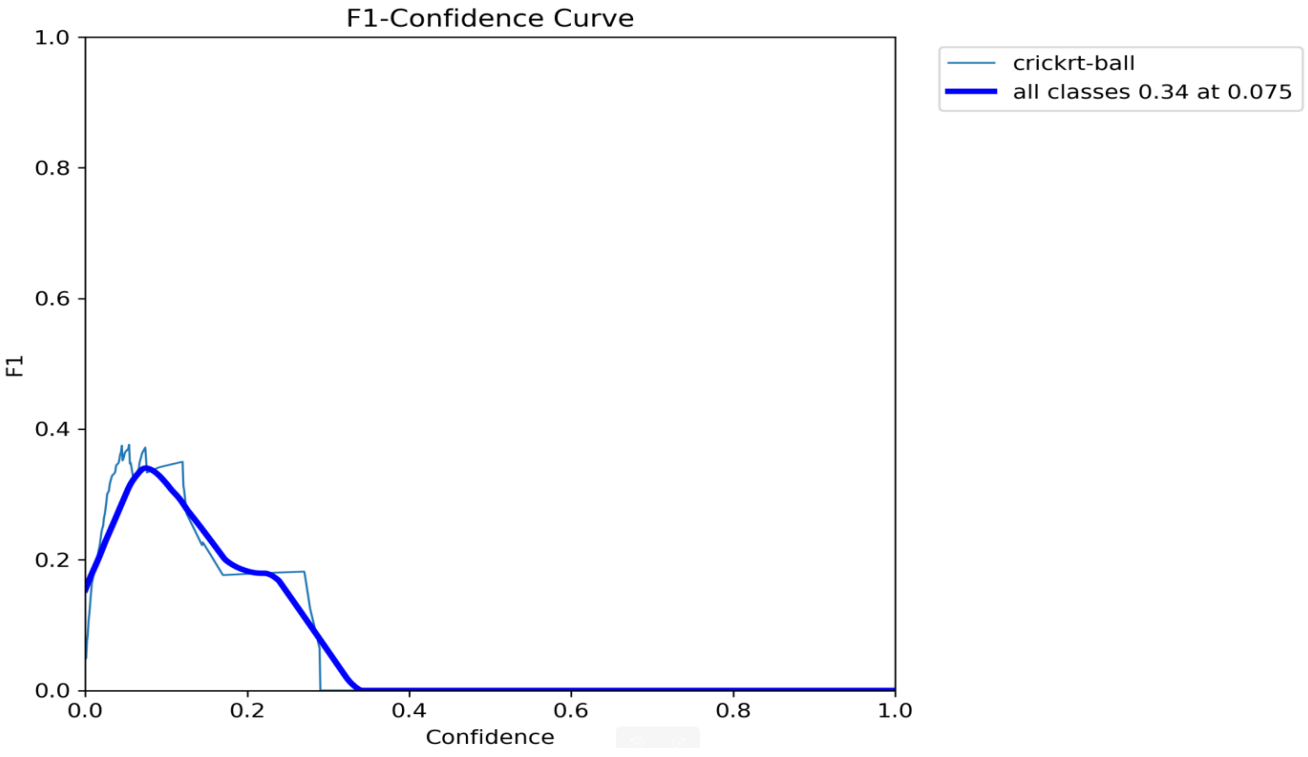
It works well with custom datasets

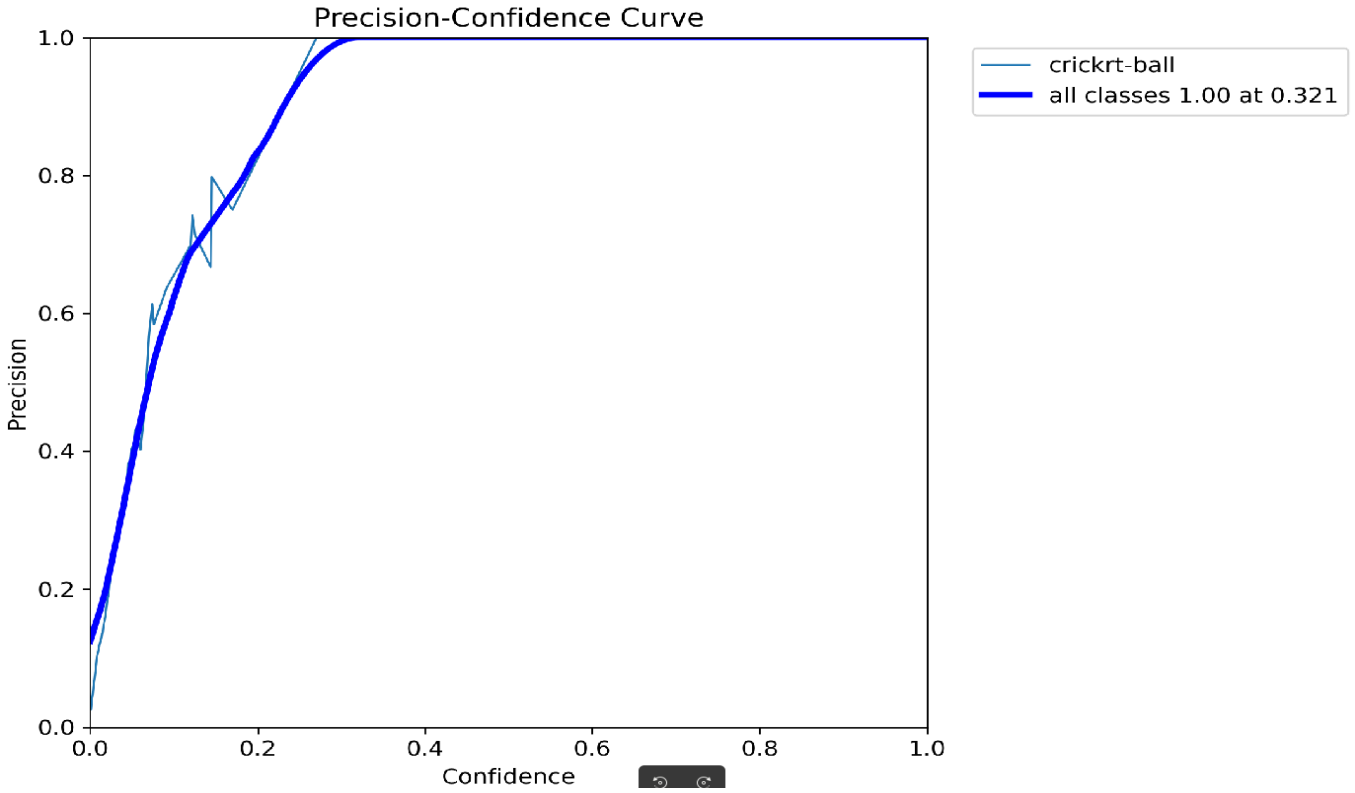
Training : These are the settings I used while training the model:

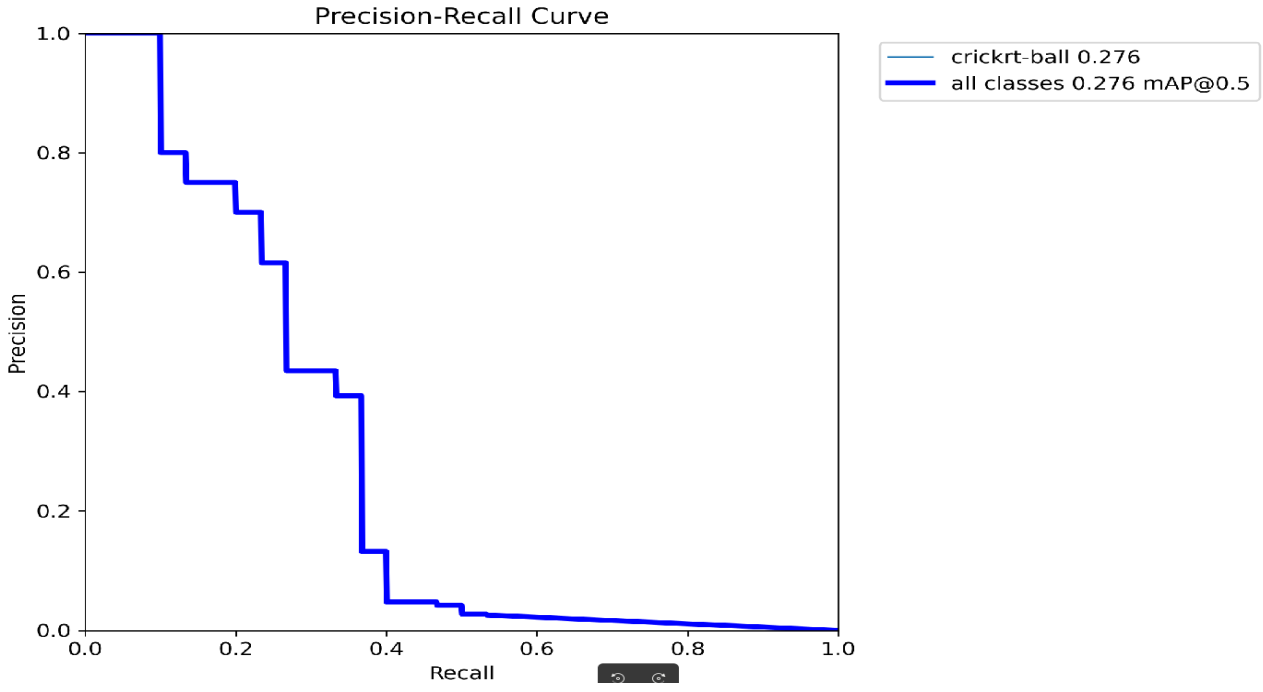
Epochs: 50
Image Size: 640
Batch Size: 16
Device: GPU
Optimizer: Default YOLO optimizer (Adam)

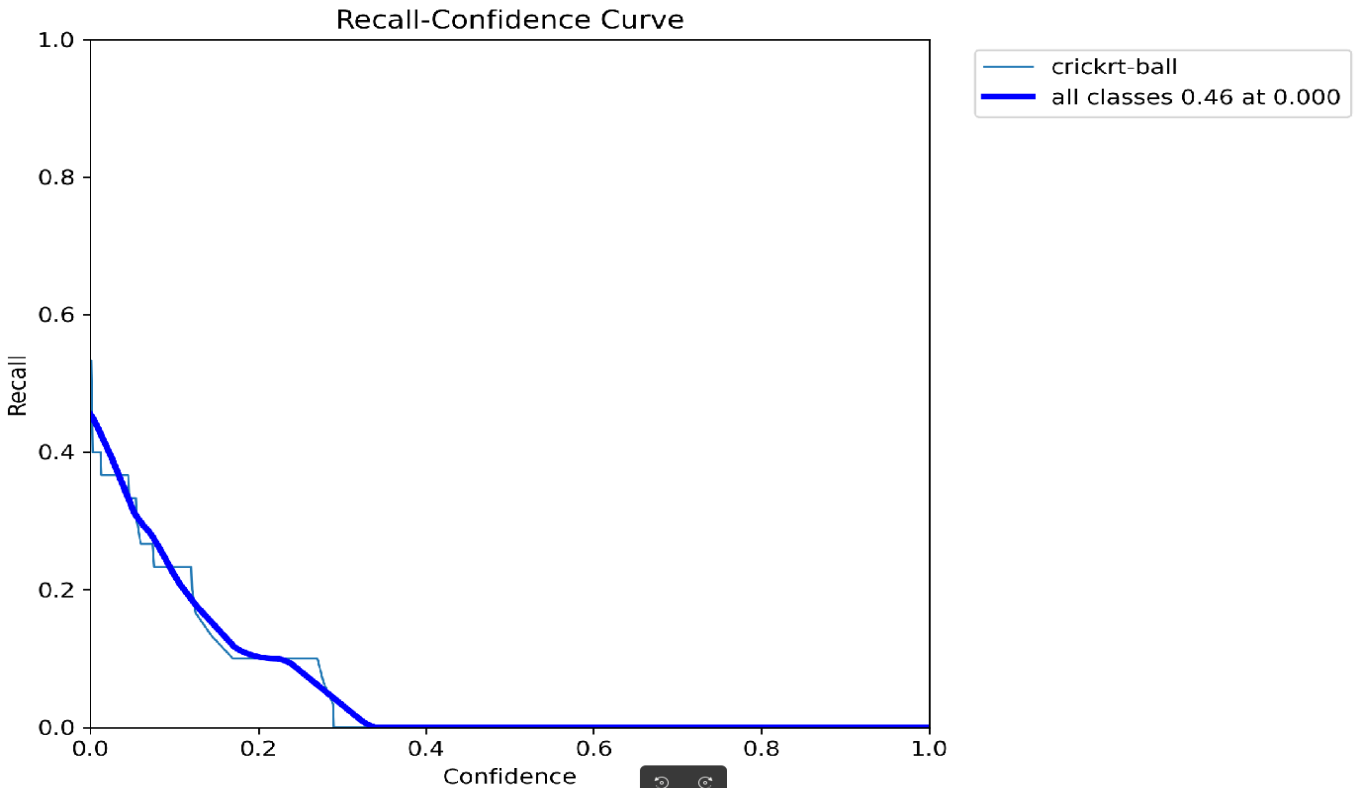
Model Evaluation: YOLO automatically generates evaluation metrics:

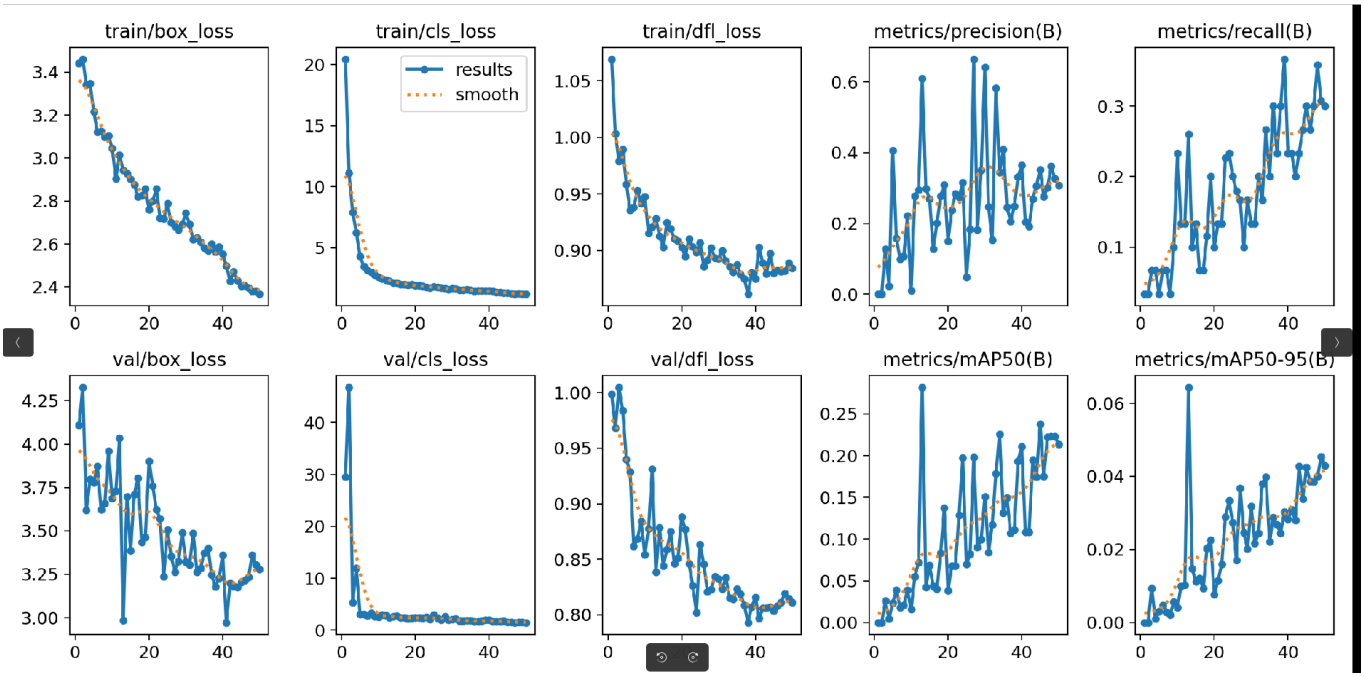
F1- Confidence curve
Precision confidence curve
Recall-confidence curve
Confusion matrix
Training batch
Result




Precision-Confidence Curve
crickrt-ball
all classes 1.00 at 0.321
Precision
Confidence
0.0
0.2
0.4
0.6
0.8
1.0


Precision-Recall Curve
crickrt-ball 0.276
all classes 0.276 mAP@0.5
Precision
Recall
0.0
0.2
0.4
0.6
0.8
1.0


Recall-Confidence Curve
crickrt-ball
all classes 0.46 at 0.000
Recall
Confidence
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0


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)