Introduction

In this project, my goal was to create a model to detect the bounding box and the number on odometers in images. Since the project requires first identifying the odometer's bounding box and then extracting the odometer number, I divided the project into three phases:

1. Dataset Gathering
2. Finding the Odometer Bounding Box
3. Extracting the Odometer Number from the Image

I separated the bounding box detection from the image-to-text phase because these are two complex tasks. Combining them into a single task could increase difficulty and require a larger dataset, longer training time, and more resources.

Dataset gathering

In this step, I explored available open-source datasets and came across the TRODO dataset, which contains over 2000 odometer images with annotations. Although approximately 2000 images may not be sufficient for a complete training process, I decided to use transfer learning with a pre-trained object detection model to address this challenge. The directory structure of the dataset is shown in Image 1.

Image 1 - TRODO dataset structure

In the annotations folder, each image is accompanied by an XML file that specifies the locations of all types of boxes found in the image. You can find this dataset here.

Finding the odometer bounding box

To detect the odometer bounding box, I tried to train a YOLO Model. As I had a labeled dataset (TRODO), my task was to train a model supervised. I divided my data into two parts, train and test having a share of 70% for the training set and 30% for the validation set.
Also, I needed to convert the annotation files to a format which is understandable for the YOYO model.
Each YOLO annotation file should have the same name as the corresponding image file but with a .txt extension.
Each line in the YOLO annotation file represents one bounding box and should follow this format:

<object-class> <x_center> <y_center> <width> <height>

where:
- <object-class> is the class index.
- <x_center>, <y_center>, <width>, and <height> are normalized values (between 0 and 1) relative to the image dimensions.

So, the final structure would be like Image 2.

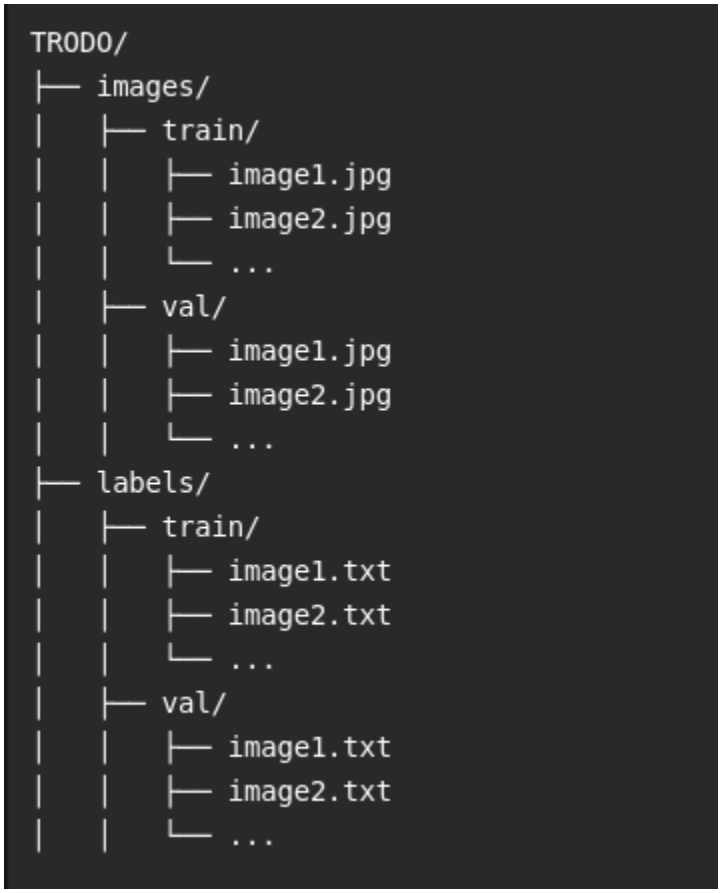

Image 2 - final dataset structure

Related codes for converting the annotation files are provided in convert_annotations_to_yolo_based_labels notebook.

Training the model

After managing the dataset, I used a YOLOv5 model for training. The reason behind my decision was that:

1. YOLOv5 is one of the leading models for object detection, known for its high accuracy and speed.
2. YOLOv5 provides pre-trained models that you can use for transfer learning, which can significantly reduce training time and improve performance on smaller datasets.

The process of training the model is coded in train_the_model notebook.
I have done the whole training process on Google Colab, as it gave me a GPU. So, some of the commands are Colab-friendly.

After achieving the model with optimal parameters, I ran the model with the best parameters on images I was given to have the images with the odometer bounding box.

Results

The final results of the model are reported in Table 1.

Metric	Precision	Recall
Train Set	94.6	96.3
Validation Set	94.2	94.8

Table 1 - Output results

Here in Image 3, you can see the Precision-Recall curve. As it shows, the curve is near the upper-right side, which sounds good. The Mean-Average Precision with at least 0.5 threshold (mAP50) of the model is 94.8%.

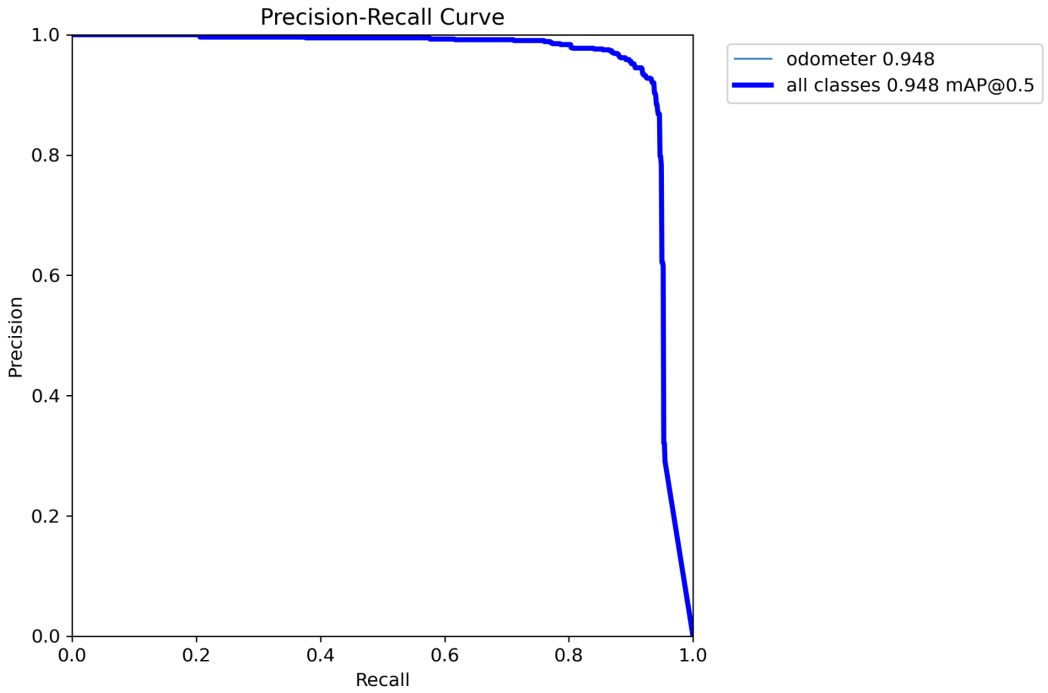

Image 3 - Precision-Recall Curve

Also, in Image 4, you can see the results on the train images, that the model has found the bounding box.

Image 4 - Result on train images

Finally, you can see the results on the test data in Image 5.

Image 5 - Test image with bbox and accuracy

By default, the YOLO model annotates images with bounding boxes and detection accuracy. However, this accuracy number can interfere with extracting the odometer reading. To address this, I modified the detect.py script to hide the bounding box and accuracy annotations. Additionally, I adjusted the script to crop the image, retaining only the area within the bounding box to focus solely on odometer extraction. These modifications significantly improved the final accuracy. This change is shown in Image 6.

Image 6 - Cropped test image without accuracy

All the test images with their bounding box are saved in yolov5/runs/detect/exp5/crops.

Extract The Odometer Number

Until now, we have odometer bounding boxes for each image. Now, it’s time to extract the odometer number.

Extract Text

Initially, I considered using Google's OCR API to extract text from images. However, even its free plan required a credit card, so I explored other options despite Google's high performance. I then tried the Tesseract library, applying extensive pre-processing to the bounding box images, but the results were still unsatisfactory. After testing other libraries like GOCR, I found EasyOCR to be a better option.
The code for applying this library to odometer images can be found in the image_to_text_extractor.ipynb notebook.
An example of text extraction is shown in Table 2.

Image	Extracted Odometer
	['294998']

Table 2 - Output of text extraction

Evaluation

For evaluation, I created an Excel file named **odometer_reading.xlsx**. This file, generated from the **image_to_text_extractor** notebook, checks whether the odometer number has been extracted correctly. My evaluation metric for correctness is less stringent than an exact match. Given the limitations of the EasyOCR library, I considered a difference of one character acceptable. Based on this evaluation, the final accuracy is **56.7%**.

Next Steps

First, one of the major challenges I faced during the project was accessing a reliable GPU. As mentioned earlier, the initial step of the project involved extracting the odometer bounding box, for which I trained the model with only 30 epochs. Training the model for a more reasonable number of epochs could lead to better results.

Second, although EasyOCR performs reasonably well, its non-deterministic behavior due to its use of Torch makes it vulnerable to runtime inconsistencies.

Third, EasyOCR returned all the numbers found in the image, but further development is needed to extract only the odometer number from the output text.