Norfolk CV DL Homework

Vibhav Nirmal

Date: 06/03/2023

Contents

1. Introduction

In this report, I present my approach to modeling a coupler detector using the provided data and the guidelines in the pdf document.

I considered two choices of object detection algorithms after looking at the dataset:

- **Faster RCNN**, which is a two-stage algorithm that first proposes potential regions of interest using a region proposal network (RPN) and then classifies and refines these proposals, and
- **YOLO**, which is a one-stage object detection algorithm that provides real-time detection capabilities and is faster than RCNN for edge device computation.

Considering that the primary objective was object detection with edge devices, I opted to utilize the YOLO algorithm.

I implemented the YOLO model in Python and PyTorch. I trained it on the data that was already in YOLO format, which made the task easier. The model takes an image as input and outputs a bounding box with a class label for the detected coupler.

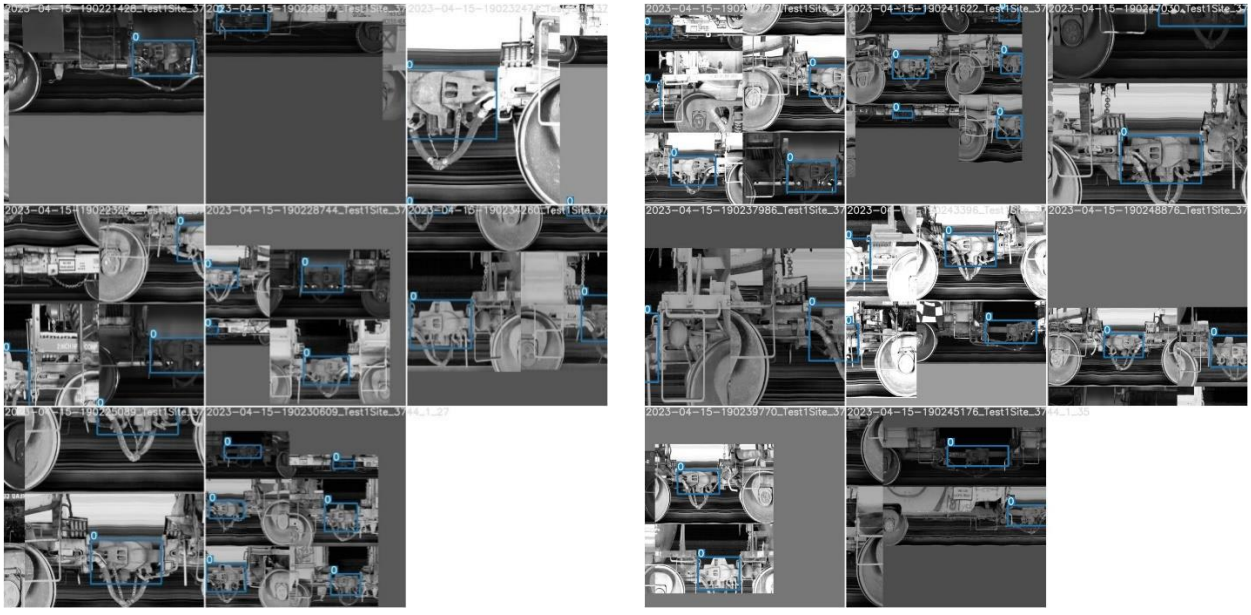
2. Model Training

Data Preparation

The dataset consisted of 200 images with bounding box annotations for the couplers. The images had different widths but the same height of 1024 pixels. I divided the dataset into **180 training images and 20 validation images** for the model training. Several **image processing parameters** were used for data augmentation. Some of them are listed below.

- **Image rotation:** 0.0 degrees (+/- deg) since all the images depicted the train and camera aligned horizontally.
- **Image translation:** A translation value of **0.2 (+/- fraction)** was applied to introduce slight variations in the object's position within the image.
- **Image scale: 0.5 (+/- gain)**, allowing for both zoom-in and zoom-out effects on the coupler region.
- **Image perspective:** Since the provided dataset showed consistent perspective across the images, the perspective parameter was set to 0.0 (+/-fraction), maintaining the original perspective without any distortion.
- **Image Flip Left-Right:** To further diversify the training data, a **probability of 0.5** was assigned for flipping the images horizontally. This augmentation helps the model learn robustly from different orientations of the couplers.

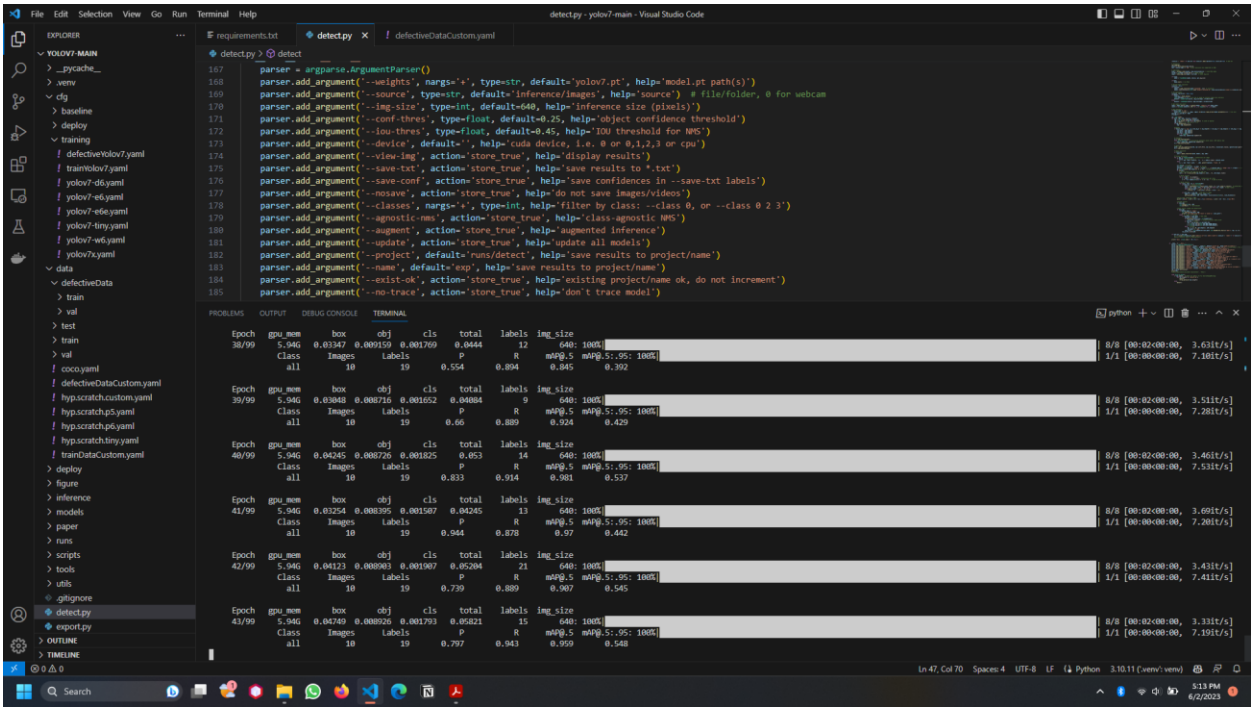
All these image processing parameters are applied during data preparation by yolov7 model.



Model configuration and Training

The YOLO model is known for its fast and efficient object detection performance, making it suitable for edge device computation. It is an **anchor-based model** that predicts potential bounding boxes and then applies **non maximal suppression** to get the best possible fit. To **optimize the model's performance**, I set the **initial learning rate** to lr0 = 0.01 and **momentum** of 0.937 to **accelerate the learning process and facilitate faster convergence**. I also applied **Weight decay** of 0.0005 to **prevent overfitting and promote generalization**.

- Model was initialized with pre-trained weights provided by yolov7 (https://github.com/WongKinYiu/yolov7/releases/download/v0.1/yolov7.pt)
- **Loss function**: YOLOv7 uses a composite loss function that consists of three components: a bounding box regression loss, a classification loss, and an objectness loss.
    - Lbox uses MSE for coordinates and dimensions, with Complete Intersection over Union as the overlap metric.
    - Lcls uses binary cross-entropy for multi-label classification.
    - Lobj uses binary cross-entropy with focal loss for bounding box confidence.
- **Optimizer:** YOLOv7 uses an **SGD (Stochastic Gradient Descent) optimizer** with a cosine annealing learning rate schedule. This helps the model avoid getting stuck in local minima and achieve better convergence.
- I trained the model for **100 epochs**, with **an input dimension of 640x640** pixels and a total **batch size of 8**.



Model Evaluation

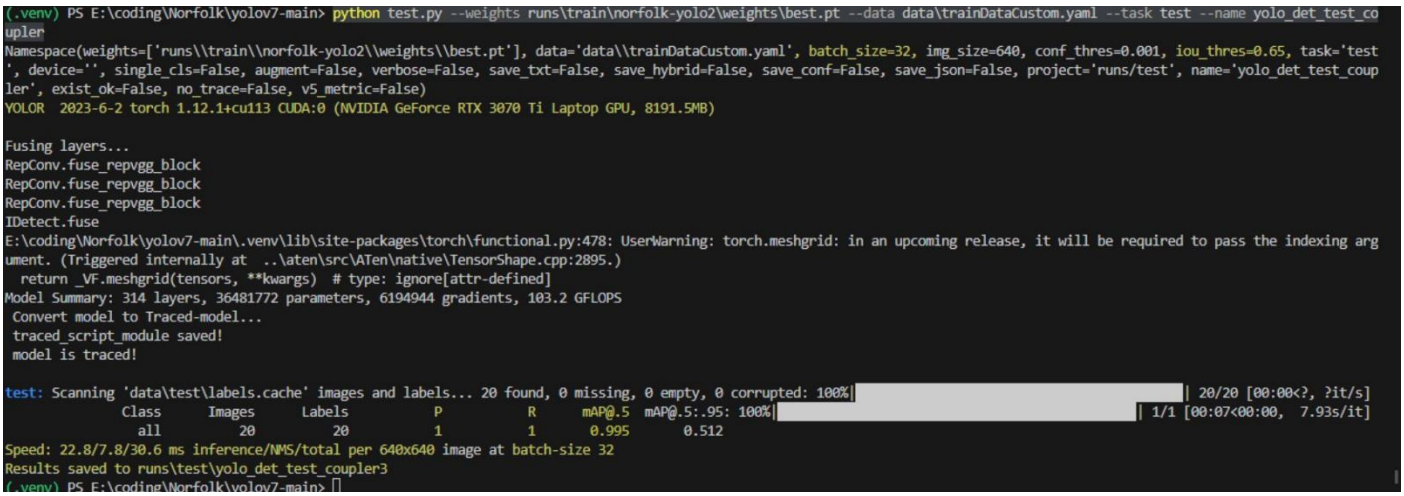
The trained model was evaluated on a separate validation dataset with 20 images to measure its performance metrics, such as precision, recall, and mean average precision (mAP)

With 180 train - 20 val split, I got the following results: (For more information I have shared the results.txt)

- The model achieved **a mean average precision (mAP) of 0.9962**, which means that it **correctly detected 99.62%** of the objects in the images.
- The model also achieved **high precision (1), recall (1) and F1-score**, which means that it **had very low (zero) false negatives and false positives.**
- The model also had **very low losses for bounding box regression (0.009411), objectness prediction (0.002513)**, and **class prediction (0)**

I did run the model on all 459 images and almost all the images had been correctly classified visually. As those images were not labeled, I labeled 20 images from that dataset (using labelImg) and evaluated the model.

Testing on **20** newly **annotated test images** produced **mAP of 0.995.**



3. Future Improvement

If I had infinite time and/or resources, I would try to improve the model performance by using:

- **more data:** More images from **different sources, conditions, and perspectives to increase the generalization** ability of the model (at least 1000 images per class)
- **different architectures**: Experiment with different **backbone, neck, and head architectures** to find the **optimal trade-off between speed and accuracy.**
- **different models: RCNN, UNet** or any other object detection/semantic segmentation models
- **different hyperparameters** and **different evaluation metrics**

Part 1: How to run code

I used **PyTorch** as the programming platform and trained the model on a **3070Ti GPU**. The code (**detectTrainCoupler.py**) and the model file (**couplerbest.pt**) are attached in this folder.

The model requires Python version 3.8 or higher to run. Use the following command to install dependencies. (Recommended to use virtualenv)

```
pip install -r requirements.txt
```

(Note that I have used torch 1.12.1 with cuda 11.3. Make sure to change according to your requirements)

To run the inference on an image and save in a file,

```
python detectTrainCoupler.py --img-path <path_to_image> --weights
<path_to_model>
```

Following command can be used to run inference on multiple images in a folder:

```
python detectTrainCoupler.py --folder-path <path_to_folder_containing_images>
--weights <path_to_model>
```

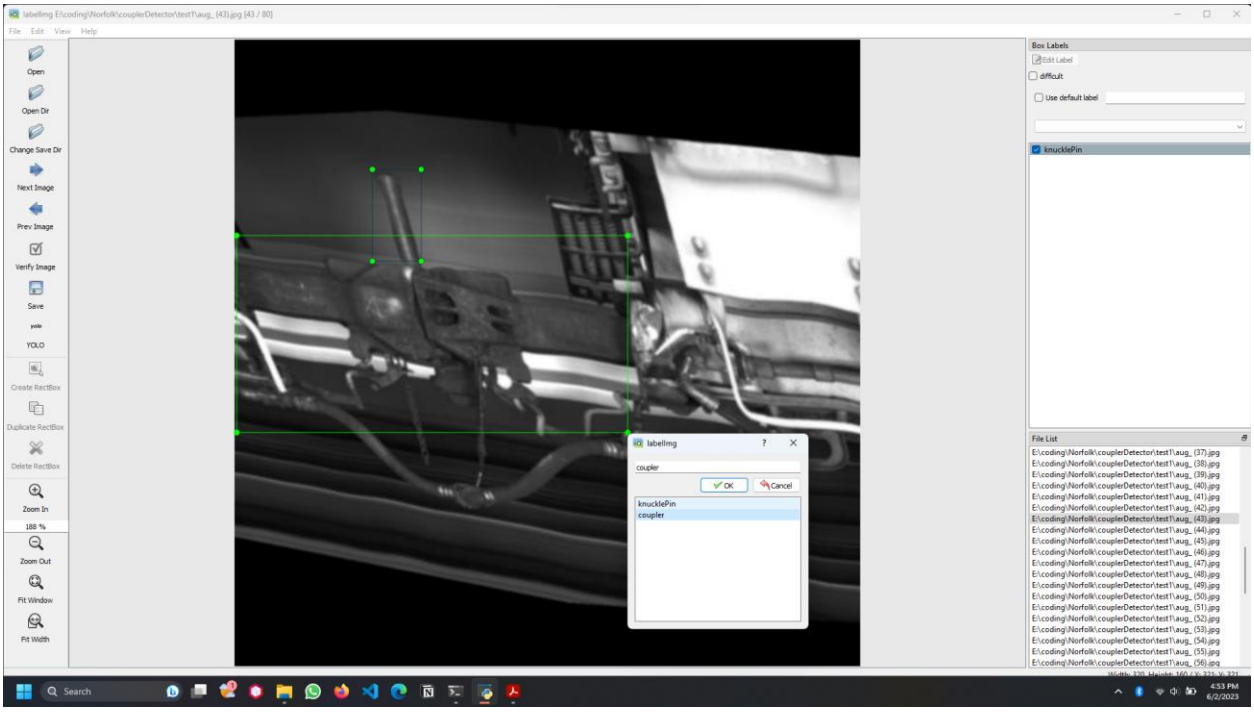
This will process all the images in the folder and save the output as images or a video file (10 fps)

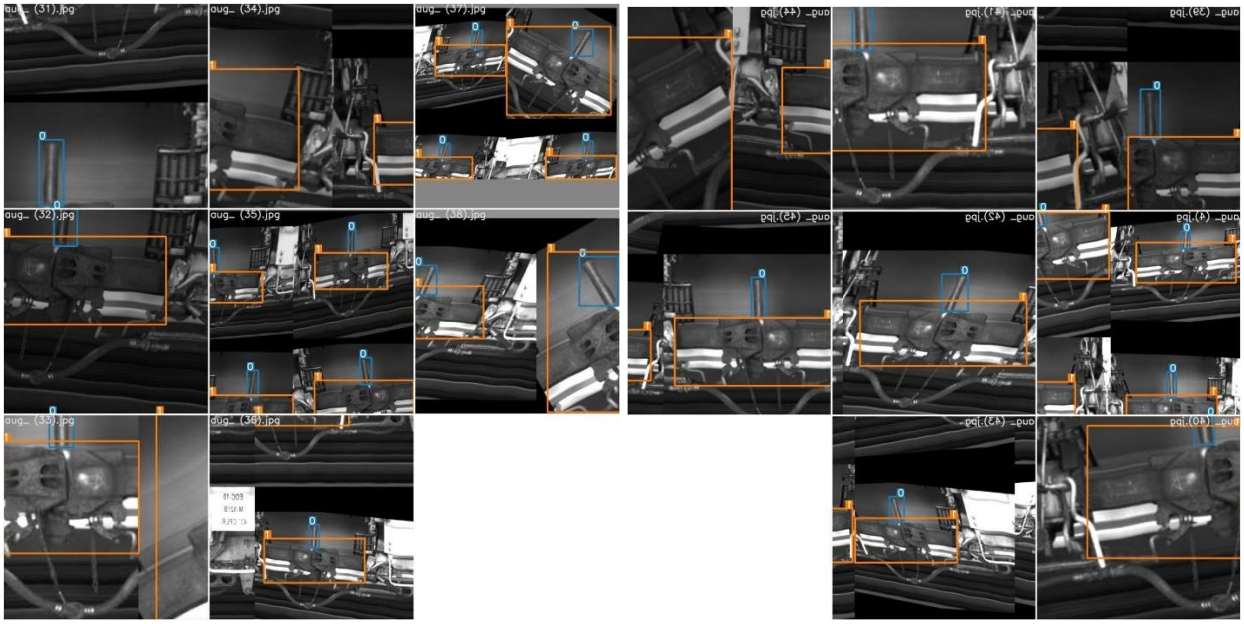
Example:

```
python detectTrainCoupler.py --weights couplerbest.pt --folder-path testImgs
--save-as-video False --conf-thres 0.4 --verbose False
```

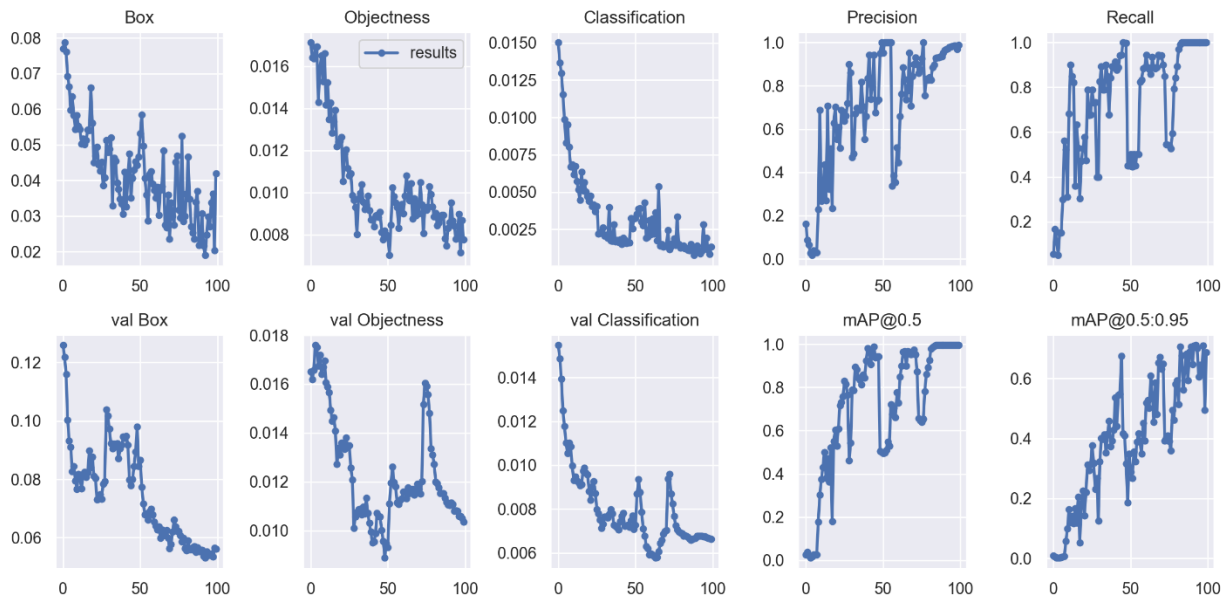
Part 2: Knuckle Pin detection

Since I only had one image to detect defects, I used some data augmentation techniques to improve the model's performance. I have augmented it by various means to create **60 training images and 10 validation images**.





However, my model was not able to predict well. I have attached the results and trained model.



To detect the defect by having a single image at hand, I can fine-tune my localizer model (couplerbest.pt) that can detect the coupler and Knuckle Pin using transfer learning and data augmentation. For yolo, this involves:

- Creating a dataset with labeled images of the defect and applying transformations such as rotation, scaling, cropping, etc. to generate more images.
- Modifying the config file to match the new classes and anchors.
- Using couplerbest.pt as the initial weights
- Training and tuning the model on the new dataset.
- Evaluating the model on a validation or test set

Data augmentation and **transfer learning** are techniques that can help overcome data scarcity by increasing the diversity of the training dataset and leveraging the knowledge of a pre-trained model. I can use yolov7 as my pre-trained model and fine-tune it on my new dataset that contains augmented images of the defect.

Also, I can use **simulated images** to train the model to make it robust and generalized.
Having different weather in simulation like rain, snow or mud can help model make better predictions over unseen images.

References:

YOLOv7 is the primary source of the whole project.
YOLOv7: The Fastest Object Detection Algorithm (2023) - viso.ai
YOLOv7: A deep dive into the current state-of-the-art for object detection | by Chris Hughes & Bernat Puig Camps | Towards Data Science