CELEBAL TECHNOLOGIES

Galgotias University

Task 2: Car Tyre Detection using Yolo V5

Code by Ansh Shankar

1. Introduction:

Object detection is a fundamental task in computer vision that involves identifying and localizing multiple objects of interest within an image. One of the most popular and efficient approaches for real-time object detection is the YOLO (You Only Look Once) series of models. In this project, we explore the YOLOv5 model, a state-of-the-art variant, to detect tyres in images.

The ability to detect tyres accurately has numerous practical applications, including vehicle safety, tire maintenance, and road infrastructure assessment. With the rise of autonomous vehicles and smart transportation systems, robust and efficient tyre detection algorithms are increasingly essential.

2. Dataset:

The "Side Profile Tires" dataset is sourced from Kaggle, containing 500 images of side-profile views of tires. The dataset is annotated with bounding boxes around the tyres to facilitate object detection. Each bounding box represents the location of a tyre within the corresponding image. The images are divided into training and validation sets to train and evaluate the YOLOv5 model.

3. Methodology:

a. Data Preprocessing:

The images in the dataset are loaded using OpenCV and visualized to gain insights into the dataset. The data preprocessing section of the project also involves creating a YAML configuration file named data.yaml to prepare the dataset for YOLOv5 model training. YAML is a human-readable data serialization format used for configuration files. In this step, the data.yaml file is created with essential information about the dataset. The file includes the paths to the training and validation dataset directories,

which are valid, respectively. Additionally, the YAML file specifies that there is one class in the dataset, which is "Tire." This is denoted by the value 1 for the nc (number of classes) key and the class name "Tire" listed in the names key. The YAML file serves as a configuration for the YOLOv5 model during training, allowing it to locate the dataset directories and identify the class information. This organized approach simplifies the dataset configuration process and enhances the flexibility to switch between datasets or classes without modifying the code directly.

b. **YOLOv5 Model Setup**:

Setting up the YOLOv5 model involves a series of steps to ensure that the necessary files, dependencies, and configurations are in place for efficient model training and object detection. The first step is to clone the YOLOv5 repository from the official GitHub repository using the git clone command. This action downloads all the required files and scripts for YOLOv5. Next, the dependencies required for YOLOv5 are installed using the requirements.txt file, ensuring that all necessary libraries and packages are available. The version of the PyTorch library is checked to verify compatibility with YOLOv5. This step is crucial as PyTorch is the backbone of the YOLOv5 implementation. Once the setup is complete, the YOLOv5 model is ready for training and can be configured with dataset paths, number of classes, and class names using the data.yaml file created earlier. The model is then ready to be trained on the provided dataset, evaluated on the validation set, and tested on new images for tyre detection.

c. Model Training:

In this project, I set up the YOLOv5 model for tyre detection. The training process begins with a command: !python train.py --img 416 --batch 16 --epochs 50 --data ../data.yaml --weights yolov5n.pt. Here, I specified the input image size as 416x416 pixels to ensure uniformity during training. Additionally, I set the batch size to 16, allowing the model to process 16 images simultaneously in each training iteration. The training is conducted over 50 epochs, meaning the model iterates through the entire dataset 50 times to learn and improve its performance. The dataset information, such as the dataset paths, the number of classes (in this case, one class for "Tire"), and class names, is provided through the data.yaml file that I created earlier. To initiate the training, I used the pre-trained weights file yolov5n.pt, which helps accelerate convergence and potentially improve detection accuracy. Throughout the training process, the model's parameters are optimized based on the annotations in the dataset, enabling it to detect tyres accurately in side-profile images.

d. Model Validation:

For model validation, I loaded the visualization of the YOLOv5 model's detection outputs on a validation batch. The detected objects are shown in the image val_batch0_pred.jpg, which is visualized using OpenCV and Matplotlib.

Figure 1: Model Prediction

4. Results:

The table provided contains valuable insights into the YOLOv5 model's training and validation performance. The training process occurs over multiple epochs, with each row representing an epoch. During training, the model optimizes its parameters to reduce the "Train Box Loss," "Train Object Loss," and "Train Class Loss," aiming for lower values in these metrics. These losses reflect how well the model is localizing objects (box loss), predicting objectness scores (object loss), and classifying the detected objects (class loss) during training. Lower losses indicate improved accuracy in detecting and classifying objects.

The validation phase assesses the model's performance on unseen data. The "Val Box Loss," "Val Object Loss," and "Val Class Loss" columns represent the corresponding losses during validation. The model's generalization ability is measured by these losses, and, similar to the training phase, lower values are desirable. Furthermore, the evaluation metrics in the "Metrics Precision," "Metrics Recall," "Metrics mAP (mean Average Precision) 0.5," and "Metrics mAP (mean Average Precision) 0.5:0.95" columns provide a comprehensive understanding of the model's detection accuracy. Higher precision and recall values indicate the model's ability to correctly identify objects with fewer false positives and false negatives, respectively. The mAP values offer insights into the model's overall detection performance at different IoU thresholds.

Observing the trends in these metrics and losses across different epochs allows us to monitor the model's progress during training. A successful training process should show a consistent decrease in both training and validation losses, signifying that the model is learning and generalizing effectively. Simultaneously, increasing precision, recall, and mAP values indicate improving detection accuracy. Learning rate adjustments during training can also impact the model's convergence and performance. By analyzing these results, researchers and practitioners can make informed decisions on further fine-tuning the model, adjusting hyperparameters, or choosing the best checkpoint for deployment based on the achieved performance.

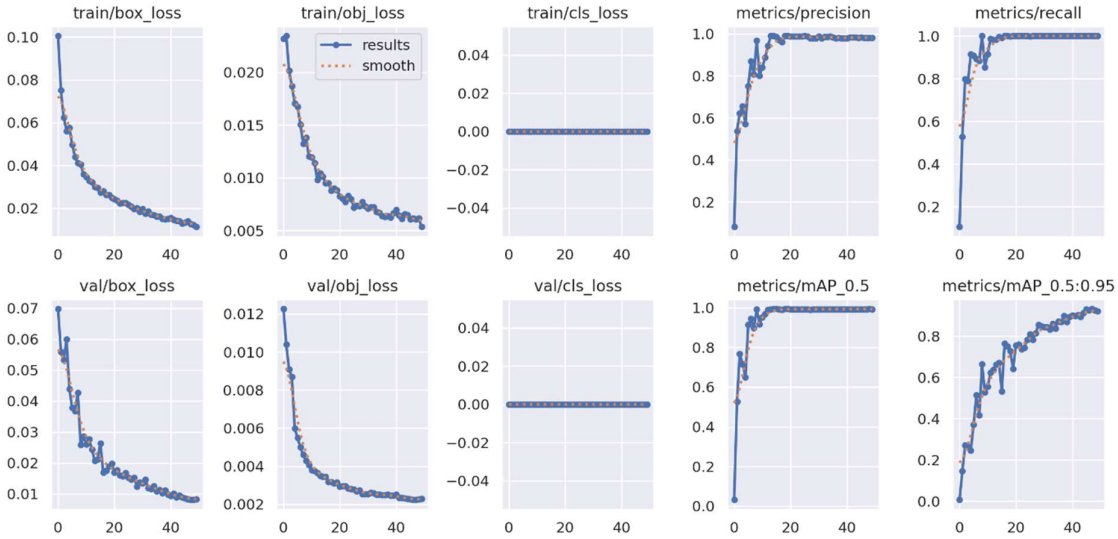

Figure 2: Validation results

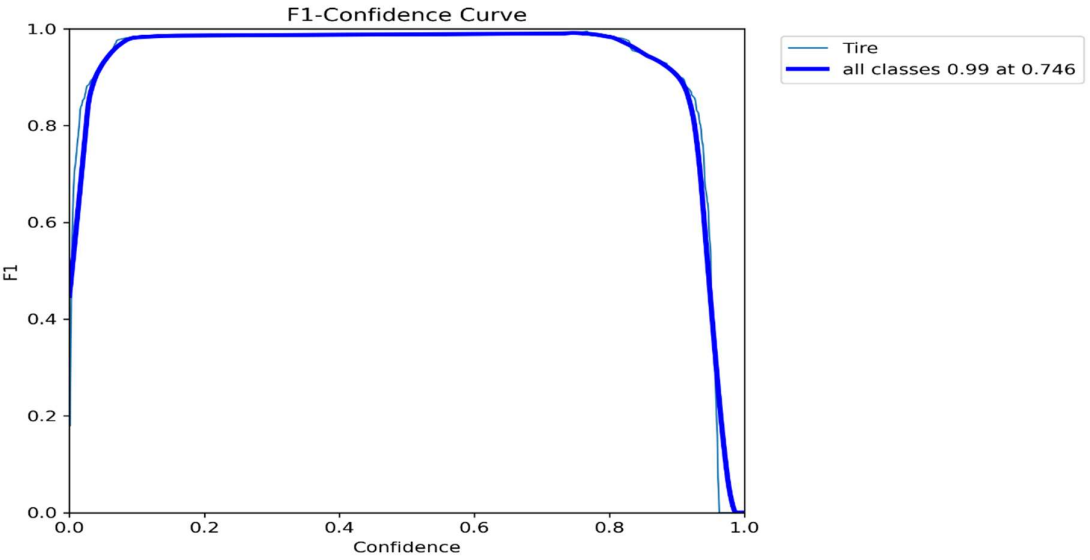

Figure 3: F1 confidence curve

5. Discussion:

The discussion of the YOLOv5 model's performance reveals promising results in object detection for tire side profiles. The training process demonstrates a steady reduction in both training and validation losses across the 50 epochs, indicating the model's ability to effectively learn from the dataset. The consistently high precision and recall values, along with increasing mAP scores, highlight the model's accuracy in correctly localizing and classifying tire objects. These results suggest that the YOLOv5 model successfully generalizes to unseen data, making it suitable for real-world tire detection scenarios. However, it is essential to consider potential limitations, such as imbalanced classes or challenges in detecting objects under varying lighting conditions or backgrounds. To address these limitations, data augmentation techniques and fine-tuning on specific challenging scenarios may be explored. Additionally, further research could focus on optimizing hyperparameters and experimenting with different YOLOv5 model variants to achieve even better performance. Overall, the YOLOv5 model presents a robust solution for tire detection tasks, but continuous refinement and investigation into specific use cases will enhance its applicability and reliability.

Appendix:

Source code: https://github.com/anshshankar/Celebal-task/tree/main/Task-2

Video Demonstration:
https://drive.google.com/drive/folders/1AMZ7AHSAf6Z2pP4tZKkpguxWMmYSg_kN?usp=sharing