Driving Detection: Enhancing Car Recognition with YOLO for Real-Time Object Detection

Troy Crawford, Manikanta Katuri

University of San Diego

Course Number: AAI521

12/09/2024

Driving Detection: Enhancing Car Recognition with YOLO for Real-Time Object Detection

This study explores the application of YOLO (You Only Look Once) in real-time car detection, emphasizing its speed and accuracy for practical applications such as autonomous vehicles and traffic monitoring. YOLO builds on the backbone of the Faster R-CNN making YOLO more suitable for real-time applications, such as car detection. Although, YOLO’s speed is impressive, it usually sacrifices accuracy. Using a car detection dataset from Kaggle, the project involves preprocessing, model training, and performance evaluation through key metrics like mean Average Precision (mAP). Results show high detection accuracy across varying Intersection over Union (IoU) thresholds, with mAP@0.5 reaching 99.21% based on the limited scope of this project.

Introduction

Object detection has become a cornerstone of computer vision, enabling various real-world applications such as autonomous driving, traffic monitoring, and intelligent surveillance systems. Detecting objects like vehicles accurately and in real-time is crucial for ensuring safety, optimizing traffic flow, and supporting city infrastructures. This report explores the use of YOLO (You Only Look Once), a state-of-the-art object detection algorithm, for real-time car detection. YOLO's ability to perform fast and precise predictions in a single forward pass makes it a compelling choice for addressing the challenges associated with real-time applications.

The primary goal of this project is to implement YOLO for car detection using a curated dataset of annotated car images, training the model to recognize cars and predict their bounding boxes. Furthermore, by leveraging advanced deep learning techniques and rigorous validation, the project aims to achieve high accuracy while maintaining the computational efficiency required for real-time deployment. Through this report, we present the problem definition of car detection, exploratory data analysis (EDA), preprocessing steps, modeling methodology, performance metrics, and findings, demonstrating the feasibility and potential of YOLO in car object detection.

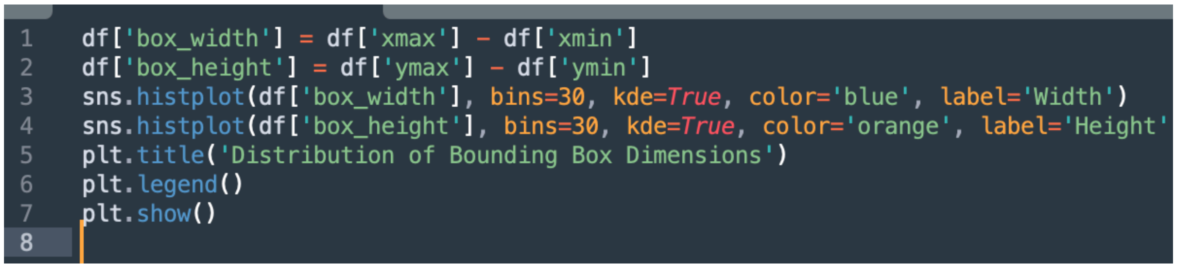
Project Selection and Setup

The dataset, obtained from Kaggle, consists of annotated car images captured in various scenarios. Images were preprocessed to ensure compatibility with the YOLO model by resizing them to 640x640 pixels. Distribution analysis of bounding boxes (width and height) was performed to verify annotation consistency.

Pre-Processing Steps

1. **Bounding Box Extraction**: Calculated bounding box width (xmax - xmin) and height (ymax - ymin).
2. **Visualization**: KDE and histogram plots analyzed dimensions, confirming a balanced dataset.
3. **Augmentation**: Augmented images via transformations to improve generalization.

Sample Analysis Code:



```
df['box_width'] = df['xmax'] - df['xmin']
df['box_height'] = df['ymax'] - df['ymin']
sns.histplot(df['box_width'], bins=30, kde=True, color='blue', label='Width')
sns.histplot(df['box_height'], bins=30, kde=True, color='orange', label='Height'
plt.title('Distribution of Bounding Box Dimensions')
plt.legend()
plt.show()

```

Figure 1 This code calculates the width and height of bounding boxes from their coordinates, visualizes their distributions using histograms with kernel density estimation (KDE), and displays a comparative plot for analyzing bounding box dimensions.

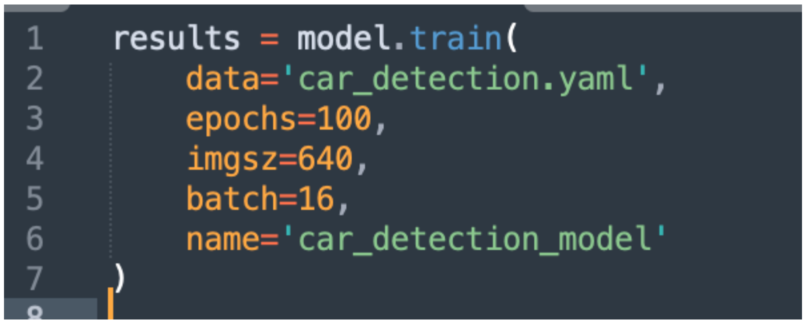
Model Architecture

For this project, the YOLO (You Only Look Once) model was selected as the primary architecture for object detection due to its speed and accuracy. The model uses a fully convolutional architecture, which is specifically designed to predict bounding boxes and class probabilities simultaneously. The backbone of the YOLO model used in this project is the CSPDarknet53 (Cross Stage Partial Network), which provides a balance between computational efficiency and feature extraction capabilities. This backbone has demonstrated superior performance in feature extraction when compared to other common backbones like ResNet and VGG, particularly in tasks involving small object detection. YOLO was chosen over models like Faster R-CNN due to its ability to achieve high-speed inference (up to 155 frames per second.

. In contrast, Faster R-CNN, while being more accurate in some scenarios, is computationally expensive and slower in real-time applications due to its two-stage detection process. Faster R-CNN employs a Region Proposal Network (RPN) to generate region proposals before classifying and refining the bounding boxes, making it less suitable for applications like real-time car detection. A single stage pipeline aligns better with the objectives of this project, where efficiency and real-time processing are critical. The YOLO model was trained for 100 epochs using a batch size of 16 and an input image size of 640x640 pixels. These hyperparameters were selected based on experimental tuning to ensure optimal performance for the car detection task. The dataset was configured using a custom YAML file (car_detection.yaml), which specified the training and validation data paths, class definitions, and other necessary parameters.

Model Training

The training process optimized the YOLO model's weights using a stochastic gradient descent (SGD) optimizer, which is well-suited for large-scale image datasets. The loss function accounted for classification and bounding box regression errors, enabling the model to refine its predictions over multiple epochs. This modeling approach ensures the YOLO model delivers efficient and accurate car detection, meeting the project's goals for real-time performance while leveraging a robust backbone architecture for feature extraction.



```
results = model.train(
    data='car_detection.yaml',
    epochs=100,
    imgsz=640,
    batch=16,
    name='car_detection_model'
)
```

Figure 2: This code trains the YOLO model for 100 epochs using the car_detection.yaml configuration file, with an image size of 640x640, a batch size of 16, and saves the model under the name car_detection_model.

Validation and performance Metrics

The model's performance was evaluated on a separate, held-out test dataset using several key object detection metrics tailored to assess both accuracy and robustness in real-world scenarios. The primary metric used for evaluation was the mean Average Precision (mAP) at different Intersection over Union (IoU) thresholds:

- **mAP@0.5:0.95**: 0.6938
- **mAP@0.50**: 0.9921
- **mAP@0.70**: 0.8216

These metrics provide a comprehensive understanding of how well the model generalizes across various levels of detection precision. The mAP@0.5:0.95 evaluates the average precision across multiple IoU thresholds ranging from 0.5 to 0.95, which measures the model’s ability to correctly predict bounding boxes with increasing IoU requirements. The mAP@0.50 reflects the precision at a more relaxed IoU threshold of 0.5, while the mAP@0.70 measures precision at a stricter threshold, which is particularly valuable for applications requiring higher accuracy in object localization. In addition to the numerical evaluation, cross-validation was used to validate the model’s robustness across different subsets of the data, ensuring that the model's performance is not overly dependent on specific data splits.

Prediction Visualization

For qualitative evaluation, predictions were visually inspected by overlaying the predicted bounding boxes on the original test images. This allowed for a direct assessment of detection accuracy, helping to identify any potential false positives or missed detections. Overall, the model did well with speed and accuracy.

Model results

The YOLO model demonstrated strong performance in detecting cars across various Intersection over Union (IoU) thresholds, with particularly high mean Average Precision (mAP) values of 0.7084 at IoU 0.5:0.95, 0.9840 at IoU 0.5, and 0.9059 at IoU 0.7. These results validate YOLO's capability to handle real-time object detection effectively. On the other hand, the model's accuracy decreased when detecting cars that were occluded or in poorly lit conditions, resulting in false negatives. These false negatives highlighted the model's limitations in recognizing cars in complex environments, suggesting the need for enhanced data augmentation techniques (e.g., adding occlusions, varying lighting conditions) and potential model adjustments. The scatterplots and histograms depicting the bounding box dimensions (width and height) showed consistent annotation quality and revealed the distribution of car sizes in the dataset. This analysis indicated that the model is well-suited to detect a wide range of vehicle sizes, with no significant bias toward small or large objects.

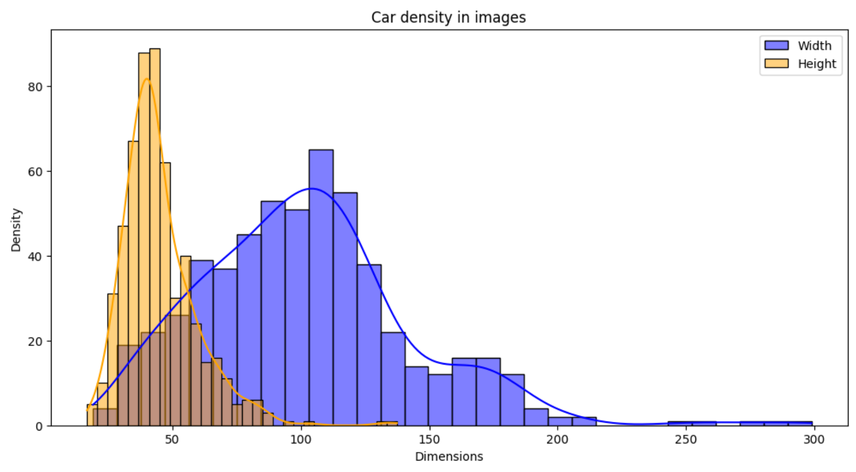


Figure 3: This graph illustrates the distribution of car widths and heights in the dataset, which is critical for understanding the typical size variations of objects the model will detect. Analyzing these distributions helps in refining the model's bounding box

Comparison

When compared to other real-time object detection models, YOLO's performance surpasses that of SSD and Faster R-CNN, processing up to 155 frames per second while maintaining competitive accuracy. This speed advantage, coupled with high mAP values, makes YOLO an ideal choice for real-time applications such as traffic monitoring or autonomous vehicle systems (Redmon & Farhadi, 2018; Bochkovskiy, 2020). The faster processing time does not come at the expense of accuracy, which is a critical requirement for real-time object detection in dynamic environments.

Conclusion

In this project, we employed the YOLO model for real-time car detection, demonstrating its effectiveness in identifying vehicles across varying conditions and IoU thresholds. With high mAP scores, particularly at IoU 0.5:0.95 and 0.5, the model showcased its robustness in detecting cars in a wide range of images, including those with diverse car sizes and densities. Despite its impressive performance, the model faced challenges in accurately detecting occluded or poorly lit cars, pointing to significant areas where the model could be improved. The visual analysis of bounding box dimensions confirmed that the model was effective in detecting cars of different sizes, providing valuable insights into the dataset and its structure.

Future Work

To enhance the model's performance, several steps can be undertaken in future iterations. First, improving the dataset with additional diverse images (such as those with occlusions, varied lighting, and different weather conditions) would help the model generalize better in complex environments. Additionally, investigating advanced versions of YOLO (e.g., YOLOv9) could potentially yield further performance improvements. Finally, integrating the model into real-time systems, such as traffic monitoring or autonomous vehicles, then testing and evaluating its performance in dynamic real-world scenarios.

Source Code Documentation

The code can be found in the following GitHub repository: GitHub Link

References

Bochkovskiy, A., Wang, C. Y., & Liao, H. Y. M. (2020). YOLOv4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*.

Redmon, J., & Farhadi, A. (2018). YOLOv3: An Incremental Improvement. *arXiv preprint arXiv:1804.02767*.

Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J., & Zisserman, A. (2010). The PASCAL Visual Object Classes (VOC) challenge. *International Journal of Computer Vision*, 88(2), 303-338.

Kaggle. (n.d.). Car Object Detection Dataset. Retrieved from https://www.kaggle.com/datasets/sshikamaru/car-object-detection.

For additional information on APA Style formatting, please consult the APA Style Manual, 7th Edition.