Real-Time Object Recognition in Urban Environments Using YOLO

Hsiaochen Huang
hsiaochen.huang@sofia.edu

Jiacheng Weng
jiacheng.weng@sofia.edu

Abstract

For the past few decades, full automation has become an ultimate goal that people want to achieve, and the development of autonomous vehicles (AVs) is one of the important progress in our lives. Understanding the surrounding and making decisions are crucial concepts for AVs. We deeply believe that the computer vision techniques can help us solve these problems.

In this project, we focus on the critical task of real-time object recognition in urban environments using the You Only Look Once (YOLO) model. By leveraging YOLO's speed and accuracy, we aim to enhance the perception capabilities of AVs, enabling them to effectively detect and classify objects such as pedestrians, vehicles, cyclists, and traffic signs.

1. Introduction

Object recognition is a critical task in computer vision, focusing on identifying and localizing objects within an image. Unlike scene segmentation, which involves labeling each pixel according to the object it belongs to, object detection aims to find bounding boxes around objects of interest. In this project, we try to make class predictions and localization, which can label each bounding box with the class of its enclosing object. The prediction provides not only the classes but also additional information regarding the spatial location of those classes.

Moreover, object recognition plays a very important role in the context of autonomous driving. The use case often requires captioning on a video stream, but this is not our focus in this project and we will evaluate our system in an offline setting. In this experiment, we started with **You Only Look Once (YOLO)** model which was first proposed by Redmon *et al.* [1]. We use **Intersection over Union (IoU)** and **Mean Average Precision (mAP)** metrics (we will mention in 4.1) as the benchmark to measure our performance because it is easy to compute and provides an intuitive approximation of the performance. The best results we achieved are ?% accuracy on training data and ?% on validation data.

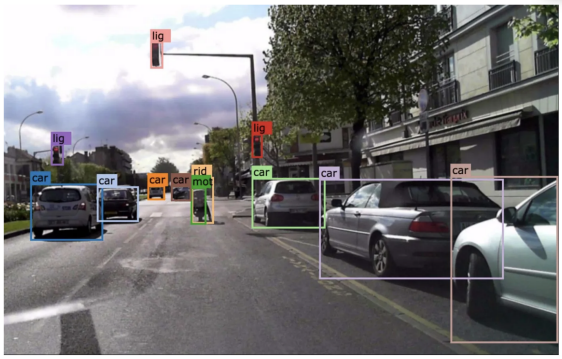

Figure 1: Example of urban scene object recognition [3]

2. Experiments

2.1. Dataset

The model will be trained and evaluated with Berkeley DeepDrive 100K (BDD100K) dataset[4]. This large-scale dataset contains 100,000 images collected from more than 50,000 rides covering New York, San Francisco Bay Area, and other regions. The dataset contains diverse scene types such as city streets, residential areas, and highways. Furthermore, the videos were recorded in diverse weather conditions at different times of the day.

The dataset is split into training (70K), validation (10K), and testing (20K) sets. Each image is in 1280x720 resolution and has a JSON label file annotated for image classification, detection, and segmentation tasks. However, we only use **Detection 2020 Labels** which contains multiple object annotations with a total of 10 categories Fig.2 and corresponding bounding boxes.

2.1.1 Dataset exploration

The training dataset provides **jpg** images and a JSON label file. The BDD100K dataset represents bounding box coordinates using the format $[X_{min}, Y_{min}, X_{max}, Y_{max}]$, where:

- (X_{min}, Y_{min}) are the coordinates of the top-left corner of the bounding box.

category_id	category_name
1	pedestrian
2	rider
3	car
4	truck
5	bus
6	train
7	motorcycle
8	bicycle
9	traffic light
10	traffic sign

Figure 2: Categories in label data

- (X_{max}, Y_{max}) are the coordinates of the bottom-right corner of the bounding box.

In order to feed the dataset into the YOLO model, we need to parse the JSON label and convert bounding box coordinates to YOLO format by normalizing them relative to the width and height of the image. YOLO format requires bounding box coordinates to be represented as normalized values between 0 and 1 with the format: $[C_{id}, X_{center}, Y_{center}, W_{norm}, H_{norm}]$, where we calculate the center coordinates and normalize width/height using the following formulas:

- $X_{center} = \frac{X_{min} + X_{max}}{2 * W}$
- $Y_{center} = \frac{Y_{min} + Y_{max}}{2 * H}$
- $W_{norm} = \frac{X_{max} - X_{min}}{W}$
- $H_{norm} = \frac{Y_{max} - Y_{min}}{H}$

2.2. Platform

Since the availability of powerful Graphics Processing Units (GPUs) will be a crucial key to train a deep convolutional neural network. We use the compute engine with one NVIDIA® RTX® **3070** GPU. Note that the memory size of 3070 GPU is 8GB and the system memory size is 32GB, so normally it has ability to accommodate most of the implementation in deep convolutional architectures. The training speed also adjusts the branch critical factor to the success in deep learning. However, with limited resources and high-resolution training images, 8 GB is still not enough. Therefore, we have to turn down the batch size, such as the maximum batch size is 16.

To build networks, we choose to use **Pytorch** as our library due to the following benefits. First, Pytorch uses dynamic computation graphs which means the graph is built on-the-fly as operations are executed. It also offers more flexibility to modify models and experiment more easily. Last but not least, Pytorch has a strong community in the research domain, and many cutting-edge research papers and projects use PyTorch. Its ecosystem includes libraries like torchvision for computer vision, and many pre-trained models are available.

2.3. Model - YOLO

YOLO was first proposed by Redmon *et al.* [1] and designed for real-time object detection, framing the task as a single regression problem to directly predict bounding boxes and class probabilities from full images in one evaluation. In this experiment, we choose to use YOLO version 8 populated in 2023 3. The architecture comprises a backbone network, neck structure, and detection head, all designed to enhance object detection performance. The backbone network is based on CSPDarknet53, which employs a Cross-Stage Partial (CSP) connection to improve information and gradient flow during training. The neck structure uses a Path Aggregation Network (PANet) to facilitate information flow across different spatial resolutions, allowing the model to effectively capture multi-scale features. The detection head incorporates dynamic anchor assignment and a novel IoU (Intersection over Union) loss function, resulting in more accurate bounding box predictions and better handling of overlapping objects.

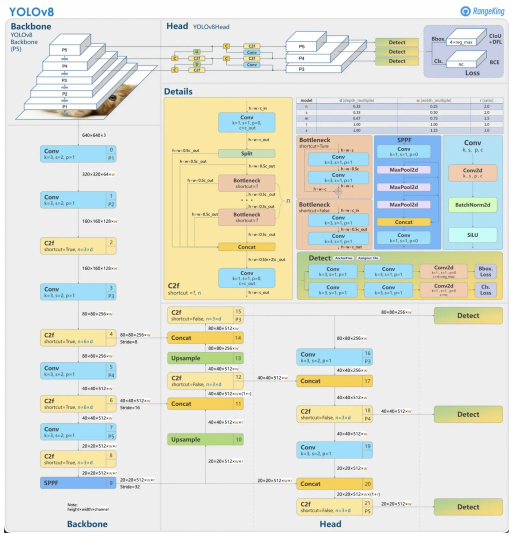

Figure 3: Architecture of YOLOv8 [2]

YOLOv8 boasts several architectural updates targeted at boosting both accuracy and speed. These include:

- Stem tweak: Reducing the first convolutional kernel size from 6×6 to 3×3 for efficient image abstraction which facilitates faster inference speed.
- Backbone bottleneck tweak: Upscaling the first convolutional kernel in the bottleneck area from 1×1 to 3×3 for better feature extraction.
- Backbone building block swap: Replacing the C3 block from YOLOv5 with a new, more efficient design.
- Anchor-free head: Implementing a novel anchor-free head for object detection, eliminating the need for predefined anchor boxes which contributes to higher accuracy compared to earlier versions
- New loss function: Utilizing a modified loss function focusing on both bounding box location and classification confidence.

YOLOv8 achieves different variants representing different scales of the model, catering to different resource constraints and performance needs. We will be using "Nano" and "Xlarge" variants , Fig. 4, in this project for comparison.

Model Variant	YOLOv8 Nano	YOLOv8 XLarge
Model Size	Small	Large
Parameter (M)	3.2	68.2
FLOPs (B)	8.7	257.8
Speed	Very fast	Slower
Accuracy	Moderate	High
mAP 50-95	37.3	53.9
Resource Requirement	Low	High
Use Case	Mobile, Edge devices, IoT	High-end research, industrial

Figure 4: Summary comparison of YOLOv8 Nano and Xlarge

3. Result

3.1. Evaluation Matrix

Here, we calculate the Intersection over Union(IoU) to support two metrics, Precision and Mean Average Precision(mAP), which measure the performance of our results. IoU measures the overlap between the predicted bounding boxes and the ground truth bounding boxes. It is calculated as the ratio of the intersection area to the union area of the two bounding boxes. A threshold IoU value is set to 0.5 which is used to determine whether a detection is considered correct or not.

Precision presents the ratio of true positive predictions to the total number of positive predictions made by the model.

- **Precision** : $\frac{TruePositive(TP)}{TruePositive(TP) + FalsePositive(FP)}$

mAP is the mean of the average precision for all classes. It gives an overall performance measure of the model across different object classes.

- **mAP** : $\frac{1}{N}\sum_{i}^{N} AP_i$, where AP_i is the average precision for class i

3.2. Comparison

The dataset contains 70,000 training and 10,000 validation images. YOLO Nano was trained in 60 epoch with batch size 16 and YOLO XLarge was trained in 20 epoch with batch size 8. We applied a lambda learning rate(LR) scheduler for both models to adjust the learning rate during training to improve performance and convergence.

- LR_e : $\frac{1-e}{E}(1 - LR_i) + LR_i$, where e is the current epoch, E is the total epoches and initial learning rate $LR_i = 0.01$

We compare the results of YOLO Nano and YOLO XLarge, the loss over the training process is shown in Fig.5. The performance is shown in Fig.6 as the precision and Fig.7 as the mAP. As our experiments, YOLO XLarge with more parameters has the best precision (0.71) and mAP(30.8%). YOLO Nano with less parameters has worse precision(0.67) and mAP (24.7%).

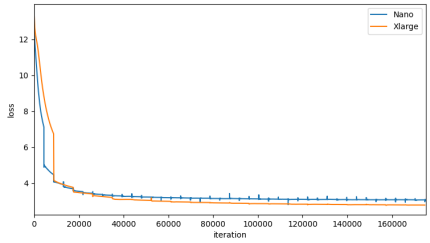

Figure 5: Loss Comparison

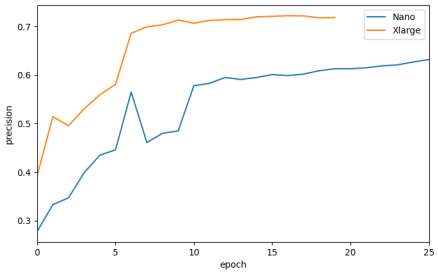

Figure 6: Precision Comparison

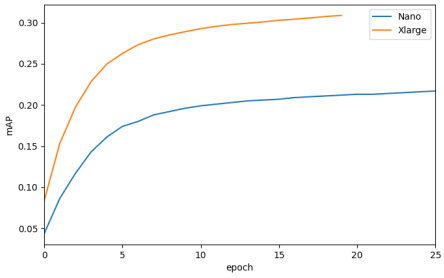

Figure 7: mAP Comparison

In Fig.8, we show the prediction results from test dataset. The first row is the result of YOLO Nano, the second row is the result of YOLO XLarge. The YOLO Nano model is missing some object detections, but the YOLO XLarge model accurately identifies those objects.

Figure 8: Predictions Comparison

4. Future Work

Due to the limitation of time and resource, we spent much time on training process and did not able to find the proper value of parameters. Hence, we hope to complete the following works in the future:

- Upgrade hardware by increasing the efficiency and memory of GPU
- Achieve real-time segmentation
- Tune parameters to find the best performance
- Try more classes or use a different dataset
- Explore different models

References

[1] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016.

[2] J. Solawetz and Francesco. What is yolov8? the ultimate guide. 2023, Jan 11. Accessed on 2024.

[3] C. Wang, I. Yeh, and H. M. Liao. Yolov9: Learning what you want to learn using programmable gradient information. *CoRR*, abs/2402.13616, 2024.

[4] F. Yu, H. Chen, X. Wang, W. Xian, Y. Chen, F. Liu, V. Madhavan, and T. Darrell. Bdd100k: A diverse driving dataset for heterogeneous multitask learning. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2633–2642, 2018.