Object Detection for Autonomous Vehicles

Module Name	**:**	Deep Learning
Intake Code	**:**	APDMF2310AI
Lecturer Name	**:**	Assoc. Prof. Dr. Raja Rajeswari
Hand in Date	**:**	4th of October, 2024
Student Name	**:**	Nurlan Amanov
Student Number	**:**	TP077526

Table of Contents

ABSTRACT

This work investigates the design and optimization of object detection systems for autonomous cars. For autonomous cars to navigate safely in challenging and dynamic situations, they need to be able to recognize objects with a high accuracy. This study examines and contrasts the advantages and disadvantages of many object detection models. Because it strikes a compromise between speed and accuracy, the YOLO model is determined to be the best appropriate for real-time detection tasks. The KITTI dataset, which contains a range of driving scenarios, is used in the study to describe the execution of YOLOv5. Important issues include managing various environmental circumstances, recognizing small or obscured objects, and striking a balance between computational efficiency and detection accuracy are covered. Metrics including precision, recall, and mAP were used to train and assess the model. It performed well overall, particularly when it came to identifying larger items like cars. However, the model could do a better job of identifying smaller or partially obscured items. The article ends with suggestions for further research, such as enhancing item detection in difficult situations and raising the model's effectiveness for practical uses.

1. INTRODUCTION AND BACKGROUND

1.1 Overview of Object Detection in Autonomous Vehicles

The most important aspect of autonomous vehicles is the development of an object detection system that identifies the environment and objects around the vehicle. Thanks to this system, the vehicle can build the right route for its journey or make the right decision during the journey, which ensures safety for such technologies. Advances in technology have enabled a leap in deep learning, which allows for highly accurate object detection. Object detection technology is a complex process, as autonomous vehicles operate in a complex environment with many different objects around them and different environmental conditions that include lighting and weather. Traditional methods have not been able to cope well with such a task, however, as mentioned earlier, with the development of technology, new models have emerged that have improved the ability to detect objects in real time and react instantly to changes in the environment.

This paper reviews a framework that will help in understanding the operation of deep learning models that are used to develop an object detection system in autonomous vehicles.

1.2 Role of Deep Learning in Object Detection

Some of the commonly used deep learning algorithms are You Only Look Once (YOLO), Single Shot Multibox Detector (SSD) and Faster R-CNN. These algorithms use CNNs for image processing, which allows them to detect multiple objects in a scene and classify them into given categories. A key capability of such deep learning models is their end-to-end learning, where deep learning allows the models to learn from the data using object classification and localization, rather than manually creating features. Additionally, the ability to utilize data from sensors, like cameras, and make real-time decisions only improves the safety and capabilities of autonomous vehicles where fast decision making is important. By continually improving such models, the field of deep learning is moving towards more robust and accurate object detection, which is helping the development of autonomous vehicle technology.

1.3 Significance of Detection for Autonomous Vehicles

In autonomous driving, the vehicle must constantly monitor its environment to identify objects, both inanimate and animate, such as pedestrians and cyclists. Reaction speeds must be millisecond fast to ensure safety on the roads, not only for the owner of the autonomous vehicle,

but also for the people around them. Failures in the system can lead to negative consequences. The challenge of real-time detection is the balance between accuracy and computational efficiency. While models such as Faster R-CNN provide high detection accuracy but require huge computational cost, models such as YOLO and SSD are designed specifically for real-time tasks, where they are optimized for faster prediction with minimal degradation in accuracy. This trade-off is important in technologies such as autonomous cars.

2. DESCRIPTION OF DOMAIN KNOWLEDGE AND PROBLEM CHOSEN

2.1 Autonomous Vehicles: A Brief Overview

Autonomous cars, commonly referred to as self-driving cars, are designed to navigate and steer without human intervention using technologies such as artificial intelligence (AI), machine learning and sensor systems. In combination with the use of sensors, computing power and various complex algorithms, the car manages its movement and makes decisions on its own. The creation of such cars carries the goal of increasing safety on the roads, their use in special cases and increasing efficiency by eliminating the human factor from driving.

The autonomous vehicle technology itself has levels starting at 0 and ending at level 5, where 0 is a vehicle without automation and 5 is a vehicle with full automation. These levels have been defined by the Society of Automotive Engineers (SAE). While levels 0-2 include a driver assistance system where the driver still plays a key role, levels 3-5 increase in the degree of autonomy, where level 5, as mentioned earlier, is transport with full automation. A Level 5 autonomous vehicle can perform all driving tasks under all conditions.

At the heart of the autonomous vehicle technology itself is a perception system that analyses the environment around it. By processing data from sensors, the system detects objects, which allows it to predict potential threats on the road. However, the system not only detects objects, but also determines lanes and plans trajectories, which are some of the tasks that the system performs.

2.2 Object Detection Challenges in Dynamic Environments

One of the hard challenges for object detection is dynamic environments. In dynamic environments, such as highways or city streets, objects around such as pedestrians, vehicles or cyclists are constantly changing. Tracking such objects and their movements is very important to ensure complete safety when using autonomous vehicles. This makes the object detection process more complex, as the models themselves need to be reliable, accurate and adaptable to all conditions.

One problem that is important to mention when referring to dynamic environments is occlusion. Occlusion is explained as follows: objects in the environment may be partially or completely occluded by other objects, making it difficult for the model to identify them. Examples of occlusion are cyclists that may appear suddenly from behind a bus. In such a case,

the detection system should identify such objects to avoid negative cases.

Another issue that is a constant challenge in computer vision is changing lighting and weather conditions. Driving at night or travelling during sunset, which can cause reflections and glare, unfavorable weather conditions such as rain or fog can all affect erroneous or missed detection. Deep learning models must be able to cope with such challenges and maintain high accuracy to ensure road safety.

Do not forget about the changing speeds on the road, because every driver is unpredictable and the ability to adapt to any situation is an important aspect for the object detection system.

Thus, solving problems in a dynamic environment for object detection systems is one of the important aspects for developing a robust model. Creating a robust, adaptable and accurate model will ensure complete road safety not only for the owner of the autonomous vehicle, but also for drivers, pedestrians and other road users around the area.

2.3 Problem Statement and Objectives

To achieve success and usefulness for future works, this paper has compiled the following main objectives:

1. The main goal is to implement and optimize a model based on deep learning, which can identify and classify objects.
2. Another challenge is to find a balance between computational efficiency and detection accuracy, thereby directing the focus towards optimizing the model.
3. The final objective is to evaluate the performance of the model using appropriate datasets and tests. The results will be compared with other existing object detection models to evaluate improvements in accuracy and processing speed.

These challenges are intended to contribute to a more robust and efficient perception system for autonomous vehicles.

2.4 Contribution and Significance of This Work

The main contribution of this paper is the review of existing methods, and the development of an optimal object detection model designed for the special task of autonomous transport. The deep learning-based models that will be considered in this paper, such as YOLO

and SSD, have proven to be effective in object detection tasks. The use of these models will help in developing a system for object detection. It is also worth mentioning that this paper contributes to optimize the balance between accuracy and computational efficiency. When this balance is achieved, the developed system can become successful. The study considers modifications to existing object detection architectures, thereby reducing the latency, which will maintain a high accuracy. The system will be suitable for real-world applications where computational resources may be limited.

The work is significant to review existing work and develop a system that will help detect objects. This research and its results will help to understand the effectiveness of deep learning models and will also serve as a basis for future work in the field of object detection, particularly in autonomous vehicles. The work can be applied to various fields: for example, in areas such as robotics, video surveillance and traffic management systems, making it useful for future research in other areas.

3. JUSTIFICATION OF SELECTED DEEP LEARNING ALGORITHM

3.1 Review of Existing Object Detection Models

3.1.1 YOLO (You Only Look Once)

You Only Look Once is an object detection model that has revolutionized the field of object detection. Unlike traditional two-stage detectors, YOLO is different in that it simplifies this process to a one-stage architecture. Thereby making YOLO efficient and resulting in an accurate and fast object detection system.

The design of this model goes by partitioning the input image into a grid, where each grid cell is responsible for predicting a fixed number of bounding boxes, abjectness estimates and class probabilities. Such a design speeds up processing and reduces computational complexity, allowing YOLO to run at up to 45 frames per second (FPS) on standard GPUs.

YOLO has had several iterations, and each new iteration has addressed limitations and improved performance. YOLOv3 and YOLOv4 have improved accuracy while maintaining high detection rates. There is also YOLOv5, but it is not an official sequel, but it is still popular. It gained its popularity due to its modularity and ease of use. YOLOv5 has pre-trained models, which allows developers to adapt the model to different object detection tasks with minimal customization. However, one of the problems that appeared with earlier versions of YOLO is the difficulty in detecting small objects that were occluded and so on. To overcome such problems, later versions, such as YOLOv5 and YOLOv8, used multi-scale feature maps, because such feature maps allow the model to better detect objects of different sizes. There were also integrated anchor boxes and spatial pyramid merging, which improved the localization of small objects. Thus, a model such as, YOLOv8, proved successful, for example, in agricultural environments where objects may be small or partially covered, showing high performance in detecting diseases such as wheat powdery mildew. (Önler & Köycü, 2024). Object detection is also being tried to be integrated into environments such as traffic monitoring and agriculture. A huge number of researchers are exploring the potential of YOLO to deal with complex environments. The integration of spatial transform networks (STNs) into YOLO, known as STN-YOLO, improves detection in cluttered environments by focusing on important image regions before detection, enhancing YOLO's ability to detect small objects in complex environments. Testing on agricultural sets has shown improved robustness to various spatial transformations

(Zambre et al., 2024). POD-YOLO and YOLOX are also worth mentioning. These models are enhanced based on YOLO to improve detection in more specialized applications, for example panoramic image detection. These models utilize deformable convolutions and spatial attention mechanisms to solve unique problems where conventional object detection models are often ineffective (Zhang et al., 2024).

The evolution of YOLO from its original form to more advanced versions has allowed it to remain the leading framework for object detection. Its ability to balance speed and accuracy, and its adaptability makes it an indispensable tool in fields such as autonomous driving, agriculture, and others.

3.1.2 SSD (Single Shot Multibox Detector)

Single Shot Multibox Detector is a deep learning-based object detection model and refers to single-shot detectors designed for real-time operation. SSD is suitable for applications that require high processing speed without significant loss of accuracy.

SSD discretizes the output space of bounding boxes by placing a set of predefined default boxes with different aspect ratios and scales on different feature maps. SSD uses feature maps at different scales, allowing objects of different sizes to be processed more naturally. This approach enhances SSD's ability to detect small and large objects in the same image. One of the advantages of SSD over other models is the balance between speed and accuracy. Although YOLO is faster, SSD achieves higher accuracy by combining predictions from multiple feature maps with different resolutions. One example is a work where SSD achieved an average accuracy (mAP) of 74.3% on the PASCAL VOC 2007 test dataset at 59 frames per second (FPS) (Liu et al., 2015). The SSD architecture allows for easy integration into existing systems where object detection components are required. Due to its open-source nature, SSD has ensured its widespread adoption in various domains.

However, SSD has limitations, for example, in detecting small objects, especially in crowded environments. To solve such problems, attention mechanisms and feature fusion techniques have been introduced in SSD. The improved SSD has improved detection accuracy and handling of small objects in dense environments (Petraschka, 2023). Such modifications improve SSD's ability to maintain a balance between speed and detection accuracy.

SSD remains a popular choice for object detection, as its ability to maintain high detection rates and provide flexibility with architectural enhancements allows it to remain

relevant to this day.

3.1.3 Faster R-CNN

Faster Region-based Convolutional Neural Network is a widely used two-stage object detection model. By implementing a region proposal network (RPN) that generates high quality region proposals for object detection makes it an efficient way. RPN is tightly integrated with the underlying convolutional network, sharing functions and significantly reducing the computational cost typically associated with generating region suggestions.

One of the strengths of the Faster R-CNN is its adaptability to different fields. For example, Faster R-CNN has been applied as wood defect detection and inspection of catenary components. The introduction of modifications such as focal loss functions and soft suppression without maximum (NMS) have improved the accuracy of Faster R-CNN. In one study, improvements to the ResNet basis resulted in a 4.4% increase in mean accuracy (mAP) for wood defect detection, and a 3.6% reduction in detection time (Zou et al., 2024).

However, the two-stage nature of the Faster R-CNN means that it is slower than single-stage detectors. Although it excels in accuracy, especially in small or closed object detection tasks, its speed of output can be a limiting factor (Wu et al., 2024).

The Faster R-CNN remains a powerful and adaptable model for object detection applications where accuracy is prioritized over speed. Its ability to handle complex, small and closed objects makes it very suitable for a variety of applications.

3.2 Comparison of Models Based on Speed and Accuracy

In the field of object detection, speed and accuracy are critical factors that determine the suitability of a model for different applications. The considered object detection models offer different trade-offs between these two parameters, making each model more suitable for specific applications depending on the requirements.

YOLO, being a single-stage detector, is optimized for speed and is capable of processing images at 45-60 frames per second (FPS) on a standard GPU. For example, YOLOv4 can achieve high speeds of up to 60 frames per second while maintaining accuracy, making it ideal for real-time systems where high detection rates are important (Zambre et al., 2024). SSD strikes a balance between speed and accuracy. SSD can achieve a frame rate of about 59 frames per second, which is slightly slower than YOLO but provides higher accuracy for small objects.

Thus, it can be concluded that SSD is generally slower than YOLO but gives more accurate results in certain scenarios (Liu et al., 2015).

Accuracy is another factor to consider. The Faster R-CNN, being a two-stage detector, has high accuracy, outperforming YOLO and SSD. For example, the Faster R-CNN with ResNet-50 achieves higher accuracy, especially where small objects are to be detected. However, the Faster R-CNN loses in speed where its frame rate is between 5 and 7 frames per second (Zou et al., 2024).

If we consider the next factor, trade-offs, then YOLO is the more favored option in this case. Since there is a slight trade-off between speed and accuracy in YOLO, where speed is more valuable in a project, this makes the YOLO model as the more preferred option. The SSD is a good middle ground, and the R-CNN is more suited for tasks where accuracy is important.

Thus, it can be concluded that from the above description and facts, the most appropriate model for this paper is YOLO.

4. DISCUSSION ON METHODS AND REFERENCES

4.1 Proposed Deep Learning Model: YOLO

The choice of YOLO is due to its balance between speed and accuracy, making it suitable for dynamic autonomous driving environments. Due to the design of the model, the input image is traversed at a time, where it simultaneously detects and classifies multiple objects, thus the YOLO approach makes it an ideal solution for systems that require fast decision making, especially when talking about autonomous vehicles where it is necessary to navigate in complex and rapidly changing environments. This project uses YOLOv5, which is adapted using a convolutional neural network (CNN). The model will be trained on a large dataset called Karlsruhe Institute of Technology and Toyota Technological Institute (KITTI), which is used in the field of mobile robotics and autonomous driving. The pre-trained network will allow the model to recognize objects such as cars, pedestrians and cyclists, which can often be encountered in city and highway traffic.

As mentioned earlier, a high quality KITTI dataset will be used. This labelled dataset will allow for more accurate object detection. The model will be evaluated using standard metrics such as precision, recall, and mean average precision (mAP). The goal is to develop a model that will detect objects quickly and efficiently, ensuring that the vehicle can maintain safety and performance even in challenging environments with many moving objects.

4.2 Dataset Selection and Preprocessing

The selection of a suitable dataset and its efficient pre-processing are mandatory steps in working with a dataset. KITTI dataset has been selected for such work. This dataset is widely popular in the field of mobile robotics and autonomous driving. It consists of a huge number of scenarios that have been recorded using various sensors such as: high-resolution RGB cameras, stereo cameras with grayscale images and a 3D laser scanner.

Any dataset must go through pre-processing. Pre-processing is designed to improve the quality of the dataset to fully ensure effective performance in the training process. Pre-processing includes the following steps: image resizing, normalization, data augmentation, anchor box calculation, handling class imbalance.

Image resizing is to resize all images to a standard size. Such actions ensure that the model can process images efficiently, while retaining important features that will help in object

detection. Normalization is needed to improve the convergence of the neural network. This action helps to normalize the pixel value, where the data values are mass scaled from 0 to 1, which helps to make the model less sensitive to different lighting conditions. Data augmentation mimics different viewpoints and distortions, which helps the model to better generalize to real driving scenarios where the appearance of objects can vary significantly. YOLO uses anchor boxes to predict the location of objects, where in a situation, if these boxes are optimized, they will help improve efficiency significantly. Handling class imbalance is a fairly common problem when dealing with datasets, where some categories of objects may be in majority and others are less frequent. To address this imbalance, techniques such as over-sampling under-represented classes or applying class-aware sampling during training are used to ensure that the model performs well for all object categories.

With a quality dataset and its preprocessing, the model will be trained to detect the objects required for the job with high accuracy, ensuring its usability. Pre-processing will help maximize model performance and ensure effective training on large datasets.

4.3 Training and Evaluation Metrics

This section describes the training procedure, key hyperparameters and metrics used to evaluate the model performance. The training process begins with pre-training the model. Transfer learning is used to exploit the ability to extract features from a pre-trained model that already has knowledge of object features. Such an approach speeds up training, which reduces the need to create an extensive labelled dataset from scratch. After pre-training, the model is tuned on domain-specific data. Fine tuning allows the model to adapt to unique characteristics such as occlusions. During training, data augmentation techniques such as horizontal flipping and color change are applied to improve the model's ability to generalize to unseen data. This step is very important to improve the robustness of the model in real driving scenarios. Key hyperparameters such as learning rate, batch size and momentum are carefully tuned to ensure optimal learning performance. For YOLO, the initial learning rate is typically set low to prevent the model from overshooting during optimization. A gradual decrease in the learning rate is used to fine-tune the weights as training progresses.

Several evaluation metrics are used to assess the performance of the YOLO model:

1. Accuracy measures the number of correctly identified objects among all objects.
2. Recall is the proportion of detected objects among all objects obtained from the survey,

indicating the model's ability to capture as many relevant objects as possible.

3. mAP is one of the most widely used metrics for evaluating object detection models. The mAP calculates the average accuracy across all object categories at different intersection thresholds. This provides a comprehensive assessment of how well the model detects objects and a higher mAP score indicates that the model performs well on different object categories and detection tasks.
4. IoU is a metric that is used to quantify the overlap between the predicted bounding box and the true bounding box. Typically, a detection is considered correct if IoU exceeds a certain threshold, for example 0.5.
5. FPS is a critical metric as it determines how many frames per second a model can process.

5. MODEL IMPLEMENTATION

5.1 Data Preprocessing

The purpose of the preprocessing step is to convert the annotation format to YOLO format. This is necessary because the KITTI dataset itself provides annotations that differ from the YOLO format. Changing the annotation format is important to ensure compatibility with YOLO.

```
yolo_labels = []
for line in lines:
    parts = line.strip().split(' ')
    class_name = parts[0]

    if class_name in class_mapping:
        class_id = class_mapping[class_name]

        bbox_left = float(parts[4])
        bbox_top = float(parts[5])
        bbox_right = float(parts[6])
        bbox_bottom = float(parts[7])

        x_center = (bbox_left + bbox_right) / 2.0 / width
        y_center = (bbox_top + bbox_bottom) / 2.0 / height
        bbox_width = (bbox_right - bbox_left) / width
        bbox_height = (bbox_bottom - bbox_top) / height

        yolo_labels.append(f"{class_id} {x_center} {y_center} {bbox_width} {bbox_height}")
```

Figure 1. Convert to YOLO Format

It is also worth matching the labels to the corresponding images, because the provided KITTI dataset provides the image and labels to them in two different files.

```
for label_file in os.listdir(kitti_labels_dir):
    if label_file.endswith(".txt"):

        img_file = os.path.join(kitti_images_dir, label_file.replace( _old: ".txt",  _new: ".png"))
        yolo_label_file = os.path.join(yolo_labels_dir, label_file)

        convert_kitti_to_yolo(
            label_file=os.path.join(kitti_labels_dir, label_file),
            image_file=img_file,
            output_file=yolo_label_file
        )
```

Figure 2. Label to Image

5.2 Model Initialization and Architecture Design

One of the other crucial steps is the initialization and design of the model architecture. This section details the process of creating the dataset structure, initializing the model with

appropriate pre-trained weights and designing the architecture.

The first thing to do before training is to split the dataset. Split the dataset into 80% for training and 20% for validation (Figure 3). This division provides a reliable estimate of model performance during training, while leaving some of the data for validation to avoid over-fitting.

```
image_dir = 'data_object_image_2/training/image_2/'
label_dir = 'yolo_labels/'

output_image_train = 'dataset/images/train/'
output_image_val = 'dataset/images/val/'
output_label_train = 'dataset/labels/train/'
output_label_val = 'dataset/labels/val/'

os.makedirs(output_image_train, exist_ok=True)
os.makedirs(output_image_val, exist_ok=True)
os.makedirs(output_label_train, exist_ok=True)
os.makedirs(output_label_val, exist_ok=True)

image_files = [f for f in os.listdir(image_dir) if f.endswith('.png')]

train_images, val_images = train_test_split( *arrays: image_files, test_size=0.2, random_state=42)

def move_files(image_list, image_output_dir, label_output_dir):  2 usages
    for image_file in image_list:

        shutil.copy(os.path.join(image_dir, image_file), image_output_dir)

        label_file = image_file.replace('.png', '.txt')
        shutil.copy(os.path.join(label_dir, label_file), label_output_dir)

move_files(train_images, output_image_train, output_label_train)
move_files(val_images, output_image_val, output_label_val)
```

Figure 3. Dataset Split

The code shown in Figure 3 divides the dataset into a folder structure. The images and their corresponding YOLO tag files are moved to specific directories for training and validation using the approach shown in Figure 4.

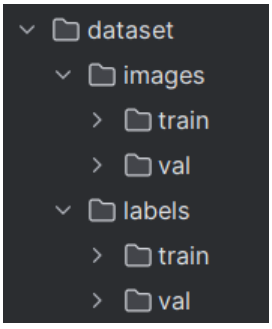

Figure 4. Directory Structure for Dataset Split

The latest YOLO models most often start training with pre-trained scales. Pre-trained models are provided with a solid foundation of learnt features for general object detection tasks. After targeting a specifically defined dataset, the model focuses on objects that are relevant to

autonomous driving, where in this case for the project it would be cars, pedestrians and cyclists. To initialize the YOLO model the pre-trained weights are downloaded from the official YOLO repository, where in this project YOLOv5 was used and ‘yolov5s.pt’ was downloaded. Due to this kind of method where pre-trained weights are used, the whole model training process is accelerated.

The YOLO architecture consists of an underlying convolutional neural network (CNN) that extracts features from the input image, followed by detection layers that predict bounding boxes and class probabilities. The key components of the YOLO architecture include:

1. Backbone network extracts hierarchical feature representations from the input image. These features are important for detecting objects at different scales and resolutions.
2. Anchor boxes represent different aspect ratios and sizes of objects. These anchor boxes are adapted to a specific data set to ensure optimal efficiency in detecting objects of interest.
3. With detection heads, the model predicts object locations and class probabilities in one forward pass, where it divides the input image into a grid and assigns grid cell boundary predictions, enabling real-time object detection.

5.3 Model Training and Evaluation

This section describes the steps involved in training the model, including the selection of hyperparameters, the training process, and the metrics used to evaluate the performance of the model. The training itself takes place in two stages, where the pre-trained model is fine-tuned first, followed by full training on the dataset used, namely KITTI.

During the initialization process, pre-trained weights are loaded, which speeds up the learning process. In fact, once the pre-trained weights are loaded, the desired dataset is used, which is called fine-tuning. Fine-tuning allows the model to specialize in detecting specifically given objects, while maintaining the general feature extraction capabilities. Data augmentation techniques are applied throughout the training process. Such practices help to increase the robustness of the model, as such is very important in the case of autonomous driving. Key hyperparameters are carefully selected to optimize the training. A batch size of 16-32 is typically used to ensure a stable gradient update, the training rate starts at a higher value to encourage fast early learning, and gradually decreases to prevent overbuilding of the model as it converges. During training, an early stop is used to prevent overshoot. If the inspection loss does not

improve within a given number of epochs, training is stopped, and the best model weights are retained. In addition, checkpoints are periodically created to preserve model states, allowing training to resume if necessary.

YOLO's loss function is a combination of three different loss types. These loss components are optimized simultaneously in the training process:

1. Localization loss measures the error in the predicted locations of bounding boxes compared to true boxes using IoU.
2. Loss of confidence determines the difference between predicted and actual objectivity estimates.
3. Class probability loss compares the predicted class probabilities with the true class labels for the detected objects.

The model's performance is evaluated using standard object detection metrics on the validation set: mAP, precision, recall, IoU and FPS.

During fine tuning, hyperparameters can be changed to help the model increase its performance. If there are any deficiencies, they should be addressed during fine tuning. It is also worth applying cross validation techniques to check and ensure that the model used generalizes well to different environments.

6. TUNING AND VALIDATION

6.1 Hyperparameter Tuning

This section looks at the key hyperparameters that are involved in training the model.

One of the first hyperparameters is the image size. This hyperparameter is one of the important ones as it affects the accuracy and speed of the computation. Finding the right image size is important because larger image sizes provide more detail, allowing the model to detect small objects more accurately, but increasing the resolution also increases the computational complexity, which slows down the learning process. This project uses an image size of 640x640 pixels. This resolution allows for a good balance between accuracy and computational speed, where it allows the model to detect objects of different sizes and maintain performance.

Another important hyperparameter is the batch size. It determines how many images are processed simultaneously during each training iteration. For example, a smaller size allows for more frequent updates, but results in noisier gradients, which can potentially slow down convergence, while a larger size results in more stable gradient updates and faster training. The larger size is the best option; however, it requires more memory. This project uses a batch size of 16, which allows us to have a balance between model performance and computational efficiency. However, it is worth realizing that this value can be changed. It just depends on the available GPU memory.

The next parameter is the number of epochs. It determines how many times the model will iterate over the entire training dataset and with more epochs, it allows the model to learn more from the data. However, it is worth realizing that this increases the risk of overfitting, where the model may perform well on the training data and poorly on the unseen data. In such a case, when validation performance does not improve, then early termination of training methods is used. This project trains the model for 50 epochs, which provides sufficient time for the model to converge without overfitting.

The learning rate controls the step size in gradient descent and is one of the most sensitive hyperparameters. If the learning rate is high, the model may simply fail to find an optimal solution, while a low learning rate may lead to sparsity in local minima. Often the learning rate is set to an initial value of 0.0010,0010,001, and after a certain number of epochs a deceleration factor is applied. Other additional hyperparameters are moment and weight decay. Momentum helps the model to maintain a consistent direction of gradient descent, speeding up

learning and avoiding oscillations. Weight attenuation helps regularize the model, preventing overfitting by penalizing large weight values. These parameters are often tuned together with the learning rate.

Finally, to run our training code we will use next command - python train.py --img 640 --batch 16 --epochs 50 --data data.yaml --weights yolov5s.pt

6.2 Model Validation and Evaluation

After training for 50 epochs, the model was validated using a set of 1,497 images from the KITTI validation set. The results are summarized below:

Table 1. Training Results

Class	Images	Instances	Precision	Recall	mAP50	mAP50-95
All	1497	7113	0.931	0.832	0.915	0.626
Car	1497	5889	0.947	0.924	0.973	0.788
Pedestrian	1497	916	0.908	0.737	0.848	0.480
Cyclist	1497	308	0.938	0.834	0.922	0.610

So based on the results let’s quickly analyze them. The model performed exceptionally well in detecting cars with a precision of 0.947 and a recall of 0.924, with an mAP50 of 97.3%. These results indicate that the model is highly reliable in identifying and localizing cars. While the pedestrian detection performance was reasonably good, the model's recall was slightly lower at 0.737, indicating that some pedestrians were missed during detection. The mAP50 of 84.8% suggests that the model is less accurate when detecting pedestrians compared to cars. This may be due to fewer pedestrians in the images. Also do not forget about occlusions, which also creates problems for detection. Cyclists were detected with a precision of 0.938 and a recall of 0.834, resulting in an mAP50 of 92.2%. The model performed relatively well in detecting cyclists, though there is room for improvement. The overall precision of 93.1% and recall of 83.2% indicate that the model is highly capable of detecting objects with minimal false positives. The model's mAP50-95 score of 62.6% shows that it performs well across a range of IoU thresholds, although more stringent IoU thresholds reveal some challenges.

7. VISUALIZATION AND CRITICAL ANALYSIS

7.1 Visualization of Model Architecture and Performance

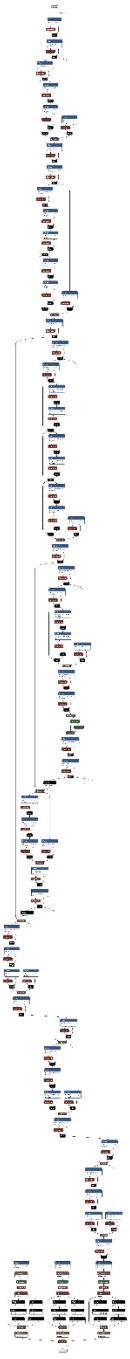

Figure 5. Model Architecture Visualization

For visualizing the YOLOv5 architecture, we will use Netron, which is specifically designed for visualizing deep learning models. To use it, you need to use the following command - python export.py --weights runs/train/exp2/weights/best.pt --img 640 --batch 1 --device cpu --include onnx. After that, go to the netron.app website and select a model. The visualization was very long and is shown in Figure 5. Architecture overview:

1. The YOLOv5 model consists of 157 layers and approximately 7 million parameters. The architecture is composed of three main components.
2. The CSPDarknet53 backbone, which is responsible for feature extraction through convolutional layers.
3. A Path Aggregation Network (PAN) that aggregates features from different scales, ensuring accurate detection of both small and large objects.
4. A detection head that predicts bounding boxes and class probabilities based on the aggregated feature maps.

The visualization of performance is generated and provided by YOLOv5 built-in visualization (Figure 6). It provides key performance metrics like loss curves, precision, recall, mAP@50, mAP@50-95 and bounding box regression.

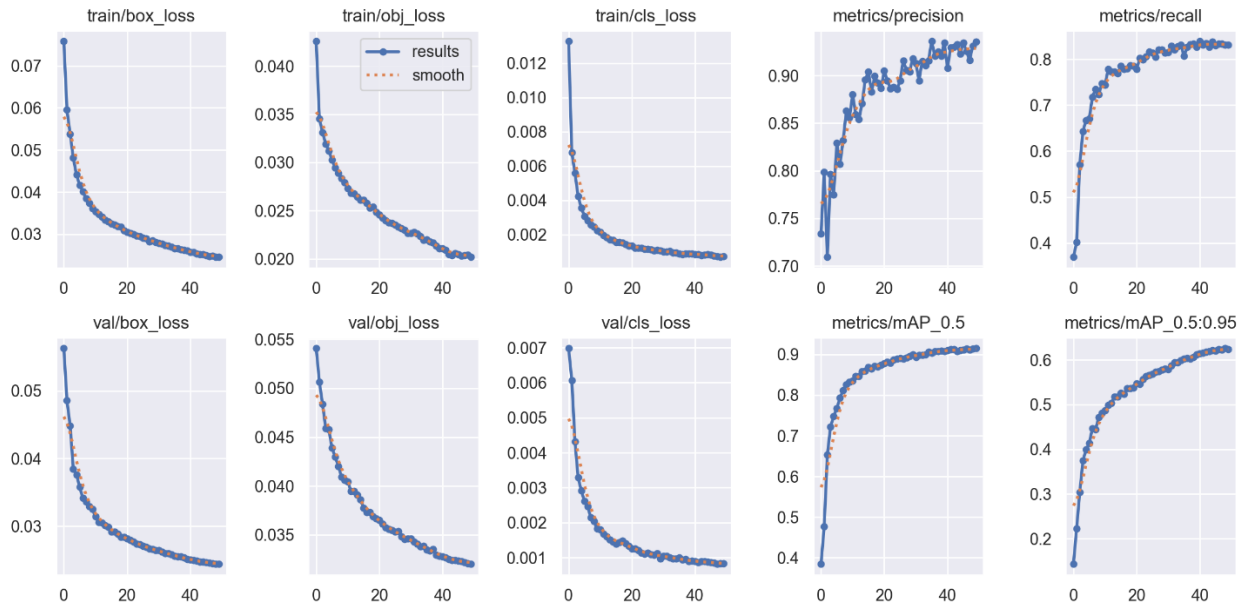

Figure 6. Performance Visualization

7.2 Critical Analysis of Model Performance

Strengths of the YOLO Model:

1. YOLO is well-suited for deployment in autonomous vehicles with an inference speed of over 30 frames per second (FPS) on modern GPUs. The model's lightweight architecture (15.8 GFLOPs) ensures efficient processing without significant computational overhead.
2. The model excels at detecting larger and more well-defined objects like cars, achieving a precision of 94.7% and a mAP@50 of 97.3%.
3. The model's strong performance across different object classes, including cars, pedestrians, and cyclists, indicates that it generalizes well to various object types.

Weaknesses and Challenges:

1. Its performance declines for smaller or partially occluded objects, such as pedestrians and cyclists. For pedestrians, the model achieved a lower mAP@50 (84.8%) and a recall of 73.7%, suggesting that the model occasionally struggles to detect individuals. Cyclist detection also exhibited similar challenges, with a mAP@50 of 92.2% and lower performance at stricter IoU thresholds.
2. Although the model achieves high accuracy at an IoU threshold of 0.5 (mAP@50 = 91.5%), its performance drops when evaluated with stricter IoU thresholds (mAP@50-95 = 62.6%). This indicates that while the model is effective at detecting objects and producing bounding boxes, the precision of these bounding boxes relative to the ground truth is less reliable.
3. Another challenge observed in the model's performance is its susceptibility to false positives in visually complex environments. In some cases, objects like trees, signs, or other background elements are mistakenly identified as objects of interest.

Areas for Improvement:

1. The model's performance on small and occluded objects could be improved by adjusting anchor boxes, fine-tuning hyperparameters related to object size, or using techniques like focal loss to emphasize harder-to-detect objects during training.
2. Further improvements in the model's localization accuracy are needed to address the drop in performance at stricter IoU thresholds. One possible solution is to enhance the bounding box regression component of the model, perhaps by employing a more advanced loss function that prioritizes accurate box prediction, such as the Generalized

Intersection over Union (GIoU) loss.

3. Techniques such as hard negative mining or more sophisticated post-processing could reduce the number of false positives in cluttered environments.

8. CONCLUSION AND FUTURE WORK

8.1 Summary of Findings

The primary findings from the model development and evaluation process are summarized as follows:

1. The YOLO model, particularly the YOLOv5 architecture, excelled in detection tasks, achieving an inference speed of 45-60 frames per second (FPS).
2. YOLO achieved high precision with a mean average precision (mAP@50) of 91.5%. The model consistently demonstrated accurate object localization and classification for well-defined and large objects in real-time environments.
3. One of the key challenges observed was the model's performance when detecting smaller or partially occluded objects, such as pedestrians and cyclists. For pedestrians, the mAP@50 was lower (84.8%), and for cyclists, it was 92.2%.
4. While the model achieved high accuracy at an IoU threshold of 0.5, its performance declined at stricter thresholds (mAP@50-95 = 62.6%), suggesting that the precision of object localization could be enhanced.

In conclusion, the YOLO model performed well in detecting objects in autonomous vehicles. However, its performance with small and enclosed objects as well as localization accuracy at higher IoU thresholds indicate potential areas for future improvement. These results provide a rationale for improving the model and exploring advances in object detection technology.

8.2 Future Work Suggestions

The following suggestions outline potential future work:

1. One of the primary areas for improvement is the detection of small and partially occluded objects, such as pedestrians and cyclists. Future work could focus on incorporating additional mechanisms to handle these challenges more effectively.
2. Future models could benefit from improved bounding box regression techniques. Introducing more sophisticated loss functions could help the model generate more precise bounding boxes.
3. Incorporating contextual information from the environment, such as road layout or traffic rules, could improve the model's ability to differentiate between relevant objects and

background noise. Context-aware object detection could reduce false positives in cluttered environments.

4. Further work on model pruning and quantization could improve its efficiency. Techniques such as weight pruning, layer reduction, or model distillation could reduce the model's size and computational complexity without sacrificing detection accuracy.
5. Future work could explore domain adaptation techniques to make the model more robust to adverse weather conditions.

REFERENCES

Aboyomi, D. D., & Daniel, C. (2023). A comparative analysis of modern object detection algorithms: YOLO vs. SSD vs. Faster R-CNN. *Information Technology Engineering Journals (ITEJ)*, *8*(2), 96-106.

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C. Y., & Berg, A. C. (2016). SSD: Single shot multibox detector. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14* (pp. 21-37). Springer International Publishing.

Önler, E., & Köycü, N. D. (2024). Wheat powdery mildew detection with YOLOv8 object detection model. *Applied Sciences*, *14*(16), 7073.

Rahmawati, L., Rustad, S., Marjuni, A., Soeleman, M. A., & Andono, P. N. (2023, September). Foggy-based object detection in video using Faster R-CNN, YOLOv3, and SSD. In *2023 International Seminar on Application for Technology of Information and Communication (iSemantic)* (pp. 412-416). IEEE.

Vilcapoma, P., Parra Meléndez, D., Fernández, A., Vásconez, I. N., Hillmann, N. C., Gatica, G., & Vásconez, J. P. (2024). Comparison of Faster R-CNN, YOLO, and SSD for third molar angle detection in dental panoramic X-rays. *Sensors*, *24*(18), 6053.

Wang, X., Li, K., Shi, B., Li, L., Lin, H., Wang, X., & Yang, J. (2023). Single shot multibox detector object detection based on attention mechanism and feature fusion. *Journal of Electronic Imaging*, *32*(2), 023032.

Wu, C., He, X., & Wu, Y. (2024). An object detection method for catenary component images based on improved Faster R-CNN. *Measurement Science and Technology*, *35*(8), 085406.

Zambre, Y., Rajkitkul, E., Mohan, A., & Peeples, J. (2024). Spatial transformer network YOLO model for agricultural object detection. *arXiv preprint arXiv:2407.21652*.

Zhang, M., Li, H., Li, Q., Zheng, M., & Dvinianina, I. (2024). POD-YOLO: YOLOX-based object detection model for panoramic image. In *Image Processing, Electronics and Computers* (pp. 236-250). IOS Press.

Zou, X., Wu, C., Liu, H., Yu, Z., & Kuang, X. (2024). An accurate object detection of wood defects using an improved Faster R-CNN model. *Wood Material Science & Engineering*, 1-7.