## Introduction

### Definition of Object Detection Task

Object detection is a computer vision task aimed to identify and track objects in images or videos. Object detection captures both the class label using bounding boxes defined by coordinates, for example from top left and bottom right corners. It includes classification to identify object classes and regression to track the changing coordinates of bounding boxes. In a regression task, it requires accurate numerical predictions, unlike simply categorical tasks that only requires classification.

### Standard Performance Metrics for Object Detection

1. Intersection over Union (IoU): IoU quantifies the overlap of two bounding boxes, the predicted box and ground truth box. It is essential in evaluating the accuracy of object detection. (Naminas, 2025)
2. Precision and Recall:

   precision focuses on accuracy for identifying object classes, recall focuses on the model's capability to find all ground truth bounding boxes. Combining precision and recall get the balance between prediction quality and quantity.

   Formula for precision: (True Positives / (True Positives + False Positives)).

   Formula for Recall: (True Positives / (True Positives + False Negatives)).

3. Average Precision (AP): computes the area under the precision recall curve, it provides a value that concludes precision and recall metrics.
4. Mean Average Precision (mAP): It is based on AP then extends by calculating the average AP of different classes. It is an important metric for current coursework performing multi class detection. It provides an overall performance metric for multi class detection.

In this coursework, I used **mAP@0.5** as the primary metric. I also used mAP@0.5:0.95, Precision, and Recall to provide a more comprehensive performance assessment on the ROAD dataset.

### Existing Object Detection Models

YOLOv5: A single-stage detector with Darknet backbone famous for real-time performance.

Faster R-CNN: A two-stage detector using a Region Proposal Network (RPN), it has high accuracy but less efficient with longer inference time.

SSD: Single Shot MultiBox Detector, using convolution feature maps in different scales to perform predictions.

DETR: Uses Transformer architecture to simplify the pipline by directly predict object bounding boxes and classes.

### Explanation of the ROAD Dataset

ROAD (Road Event Awareness Dataset) dataset was created to improve situation awareness in self-driving vehicles. Based on the Oxford RobotCar Dataset, it includes 22 long videos with 122,000 annotated frames, 1.7 million labels, and 560,000 bounding boxes. (Singh, 2023) ROAD introduces a new structure by organising road events as triplets: an active agent (pedestrian, car), the action performed (turning, crossing), and the event's location related to the vehicle (in lane, on pavement). This multi-label method allows for a full understanding of road conditions by capturing the dynamic interaction of agents, actions, and locations. ROAD puts more emphasis on context awareness than usual datasets that focus on object detection. Used with algorithms like 3D-RetinaNet, Slowfast, and YOLOv5, ROAD creates major challenges due to real world complexity and different weather conditions. ROAD is an important dataset for autonomous driving research and is expandable for future tasks like trajectory prediction and ongoing learning.

**Yolov5 Training and experiment**

**Architecture and Principles**

YOLOv5, developed by Ultralytics. It is a popular single-stage object detection model for its balance of speed and accuracy. It consists of three main components: backbone as CSPDarknet53 for feature extraction, a neck PANet for feature aggregation, and head for making prediction bounding boxes and class probabilities. YOLOv5s, is the second smallest yolov5 variant it has 7.2 million parameters, making it suitable for development on my local device, with limited computation power. The model processes images in single pass, dividing them into a grid and predicting bounding boxes and class probabilities. It is optimized by anchor-based detection and loss functions like CIoU to optimize bounding boxes.

**Hyperparameters used:**

Image Size: 640x640

Batch Size:32

Epochs: 20

Weights: Started with pre-trained yolov5s.pt

The model is trained using following command:

python3 train.py --img 640 --batch 48 --epochs 20 --data road.yml --weights yolov5s.pt --device mps --name road_yolov5

The YOLOv5s model was trained on the ROAD dataset, which includes 122,000 frames across 22 videos, annotated for 11 classes (Pedestrian, Car, Bus). The dataset was split into three parts. Training, validation, and test sets. Training was conducted on an Apple M1 Mac using the MPS backend. The dataset was separated into images and labels file, aligned with YOLO format, then changed to the right path name in road.yml.

**Hyperparameter Tuning**:

1. Different yolov5 model: yolov5s, yolov5m
2. Adjusted learning rate: 0.01, 0.02
3. Batch size: 32, 48,64
4. Image size: 640, 896

Validation mAP0.5 is used to choose the best model, saved as best.pt.

Challenges:

Try different parameters to find the sweet spot to ensure training efficiency and maintain a high accuracy. Tried to use a larger yolov5 model and larger image size, but due to computation limitation, I have to choose a smaller model and image size. It will result in a lower accuracy but achieve a faster training time.

Slow training due to some of the process goes back to CPU. I tried to implement MPS (Metal Performance Shaders) to speed up the training process. It took more than 6 hours to perform a 20 epochs training.

**Detection Results on test set**

I used following command to validate the model: python3 val.py --data road.yml --weights runs/train/road_yolov5/weights/best.pt --task test --device mps

Here is the training result

| epoch | metrics/precision | metrics/recall | metrics/mAP_0.5 | metrics/mAP_0.5:0.95 |
|---|---|---|---|---|
| 0 | 0.31051 | 0.32679 | 0.27573 | 0.13631 |
| 1 | 0.33684 | 0.50999 | 0.36371 | 0.1879 |
| 2 | 0.40425 | 0.40257 | 0.34122 | 0.16041 |
| 3 | 0.45712 | 0.43122 | 0.40899 | 0.21043 |
| 4 | 0.46017 | 0.35564 | 0.37489 | 0.19364 |
| 5 | 0.46024 | 0.48651 | 0.46081 | 0.2327 |
| 6 | 0.49169 | 0.39905 | 0.40157 | 0.21346 |
| 7 | 0.43987 | 0.36468 | 0.38851 | 0.20224 |
| 8 | 0.47031 | 0.4135 | 0.39381 | 0.21615 |
| 9 | 0.49511 | 0.3447 | 0.37382 | 0.20757 |
| 10 | 0.48868 | 0.3529 | 0.34525 | 0.17731 |
| 11 | 0.48495 | 0.36543 | 0.39627 | 0.20658 |
| 12 | 0.47631 | 0.32037 | 0.32191 | 0.17259 |
| 13 | 0.5046 | 0.32731 | 0.33997 | 0.18387 |
| 14 | 0.49748 | 0.35168 | 0.32579 | 0.15908 |
| 15 | 0.42346 | 0.39836 | 0.40702 | 0.22186 |
| 16 | 0.49316 | 0.33783 | 0.36338 | 0.20503 |
| 17 | 0.38468 | 0.32897 | 0.31599 | 0.16657 |
| 18 | 0.49827 | 0.34919 | 0.32785 | 0.16556 |
| 19 | 0.45621 | 0.33868 | 0.348 | 0.18194 |

Training losses box (from 0.070292 down to 0.028943, down 59%), objectness loss (from 0.042878 to 0.023325, down 46%), and classification loss (0.02995 to 0.0016011, down 95%) It lowered training loss consistently after each opoch . Showing decent training result with gradually decreasing training loss.

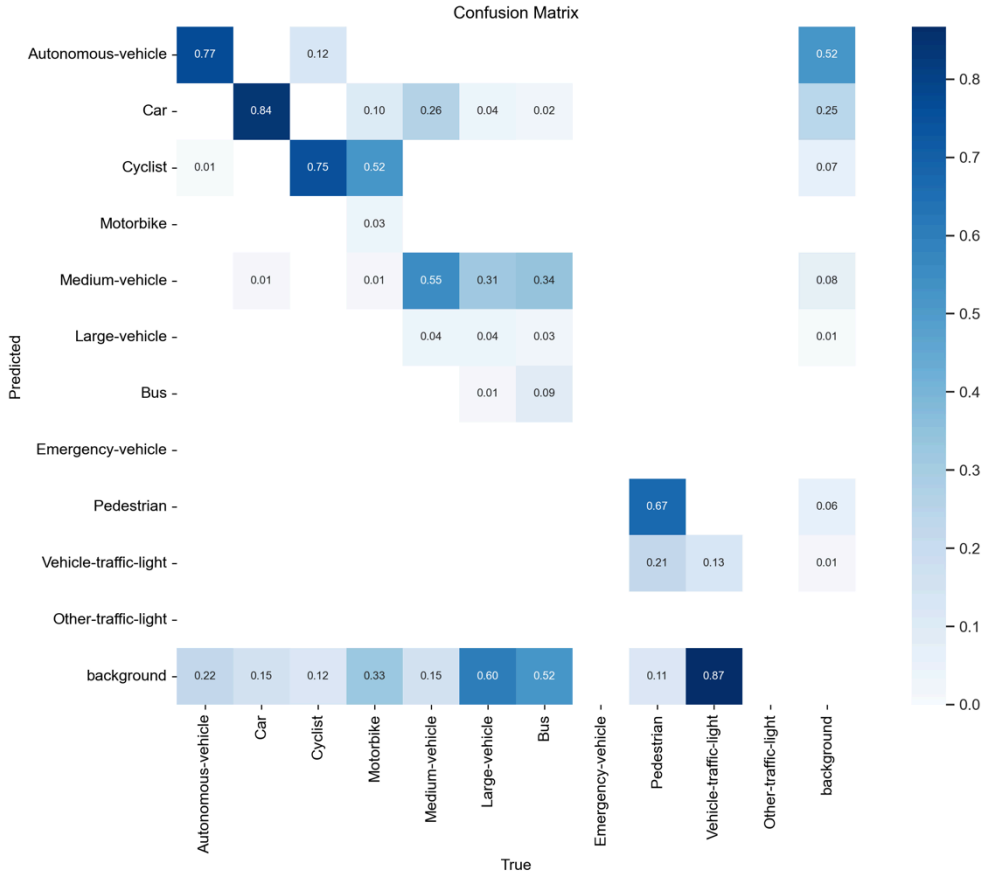
However, it is different for validation losses. Validation box loss (from 0.059522 to 0.053551) and objectness loss (0.023695 up to 0.035176). It showed no significant improvement, there is potential overfitting or difficulties in generalizing to unseen data. The mAP@0.5 peak at 0.46081 in epoch 5 but keep fluctuating after epoch 5 ending at 0.348 epoch 19. It shows a good classification performance.

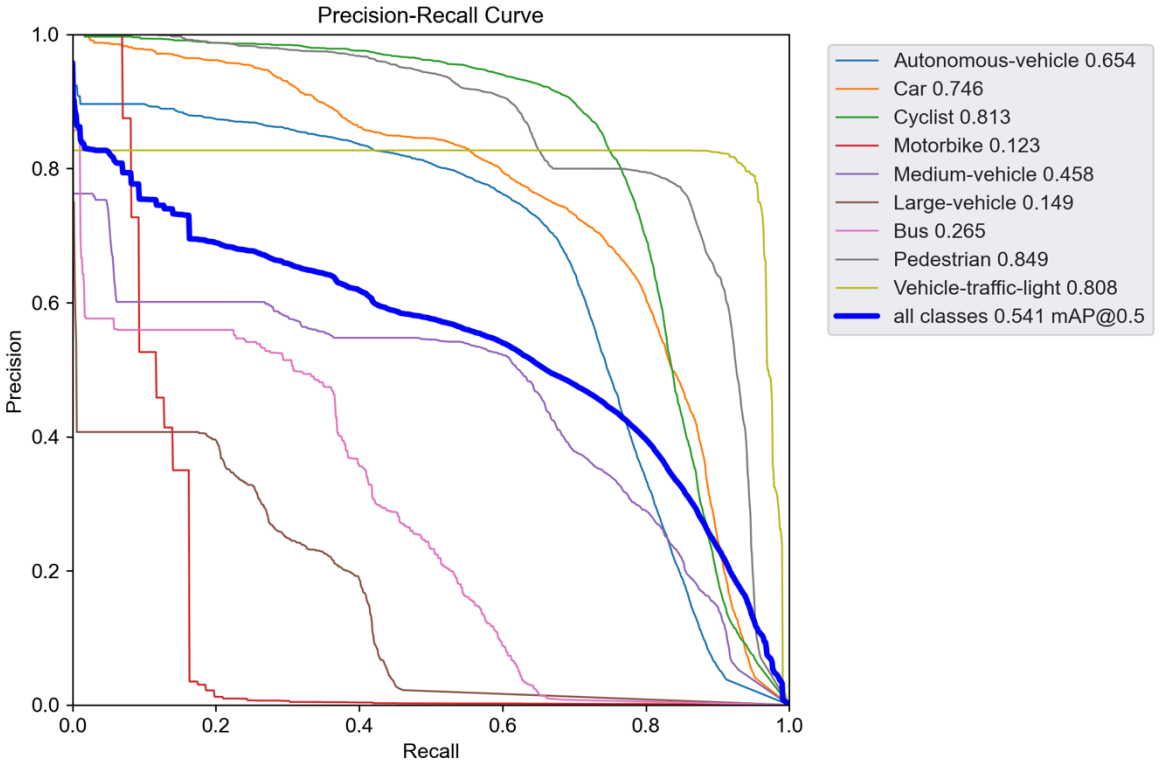
The learning rate, reduced from 0.070063 in epoch 0 to 0.00406 in epoch 19, to ensure stable convergence.

Here is the result after 20 epochs:

| **Class** | **Instances** | **Precision** | **Recall** | **mAP@0.5** | **mAP@0.5:0.95** |
|---|---|---|---|---|---|
| All | 43,720 | 0.64 | 0.534 | 0.541 | 0.249 |
| Autonomous-vehicle | 14,043 | 0.501 | 0.746 | 0.654 | 0.295 |
| Car | 7,391 | 0.513 | 0.833 | 0.746 | 0.474 |
| Cyclist | 7,363 | 0.795 | 0.767 | 0.813 | 0.422 |
| Motorbike | 86 | 1.0 | 0.0596 | 0.123 | 0.0364 |
| Medium-vehicle | 3,819 | 0.508 | 0.615 | 0.458 | 0.282 |
| Large-vehicle | 2,305 | 0.34 | 0.0572 | 0.149 | 0.0785 |
| Bus | 588 | 0.514 | 0.115 | 0.265 | 0.127 |
| Pedestrian | 6,702 | 0.81 | 0.663 | 0.849 | 0.377 |
| Vehicle-traffic-light | 1,423 | 0.779 | 0.954 | 0.808 | 0.149 |

1. Performance: mAP@0.5 of 0.541 shows a good detection accuracy, but mAP@0.5:0.95 of 0.249 suggests the model struggle to locate precise bounding boxes.
2. Performance for high instance class: Good accuracy for high instance classes. Pedestrian at 0.849 for mAP@50, Cyclist at 0.813 for mAP@50 and car at 0.746 for mAP@50
3. Performance for low instance class: Low accuracy for low instance classes. Motorbike at 0.123 for mAP@50 and Bus at 0.265 for mAP@50.






The curve illustrates mAP@0.5 of 0.541 for all classes. Precision scores vary across different class, Pedestrian is the highest at 0.849, followed by Cyclist 0.813 and Vehicle-traffic-light 0.808. However, Motorbike at 0.123 and Large-vehicle 0.149 with low precision score. It shows insufficient training examples for those specific classes, which cause class imbalance. The graph proved the model's capability in detecting classes which has high instance count like Pedestrian but struggle to detect lower instance classes like Motorbike.

**Reflection**

The YOLOv5s model performed generally well for classes with high instances. It is because of more training example helped the model to generalise better. However, low instance classes resulted low accuracy from class imbalance, In the dataset it only has 86 Motorbikes. Small object size is another challenge when using 640x640 resolution. There is potential overfitting, as validation mAP@0.5 dropped from 0.46081 in epoch 5 to 0.348 in epoch 19, but training losses continued to decrease. For future improvement, use a higher resolution like 896x896 or a larger model YOLOv5l, or implementing class weighting to reduce class imbalance.

**Experiments on Object Detection with a Model of Your Choice**

**Architecture and Principles of Faster R-CNN**

Faster R-CNN (Region-based Convolutional Neural Network) is a two-stage object detection model introduced. (Ren, 2015). It improves earlier models like R-CNN by integrating a Region Proposal Network, making it more accurate and efficient. Faster R-CNN uses a 2-stage approach. First it generates class-agnostic proposals with RPN. Then followed with Fast R-CNN, it can achieve high detection accuracy. However, requires higher computational power compared to single stage model like Yolov5. In this experiment, I used torchvision Faster R-CNN with ResNet-50, trained using ROAD dataset.

Backbone Network: ResNet-50, convert input image to produce a convolutional feature map. The feature map is generated by the last shared convolutional layer, capturing a range of features. In my experiment, ResNet-50 was trained on the COCO dataset.

Region Proposal Network (RPN): The RPN is a convolutional network that generates region proposals by sliding a small network over the feature map from the backbone. At each sliding-window location, the RPN simultaneously predicts

multiple region using bounding boxes with defined scales and aspect ratios. Top proposals are chose by Non Maximum Suppression.

Classification: The feature maps are passed to Fast R-CNN detector. It performs two tasks, classifies each region into object classes or background using softmax and draw the bounding box.

**Tuning of Model Parameters on the Validation Set**

The ROAD dataset was split into training, validation, and test sets with COCO format.

Hyperparameters:

Batch size: 4

Learning rate: 0.005, using Stochastic Gradient Decent (momentum=0.9, weight decay=0.0005).

Pytorch learning rate scheduler: step_size=3 and gamma=0.1.

Number of epochs: 20.

Images were transformed using torchvision.transforms, convert to tensors. An image loading function was implemented to load the ROAD dataset. It is to prevent crashes during training, by skipping images with no annotations during training.

After each epoch, the model was evaluated with validation set.

The initial loss was 3.2139, decreasing steadily over 20 epochs

Result after 20 epochs showed significant improvement:

mAP@0.5:0.95: 0.32

mAP@0.5: 0.57

Small objects AP: 0.27

Medium objects AP: 0.33

Large objects AP: 0.62

Detection Results on the Test Set:

mAP@0.5:0.95: 0.31

mAP@0.5: 0.56

Small objects AP: 0.26

Medium objects AP: 0.32

Large objects AP: 0.61

**Reflection on the Results:**

YOLOv5 achieved an mAP@0.5 of 0.541 on the test set.

Faster R-CNN's mAP@0.5 of 0.56 slightly higher YOLOv5, showing two-stage detectors can achieve a higher accuracy. Faster R-CNN using two-stage approach, with RPN, is more accurate in terms of detecting large objects but it missed some small objects since the bounding box is fixed in RPN, which makes it harder to detect smaller objects.

The ROAD dataset includes classes like cars, large vehicle. Faster R-CNN performs well on large objects with 0.61 AP, it can be due to these classes are large in the image. However, it didn't perform well on medium objects and small objects with 0.32 and 0.26 AP respectively.

**Reflections and Conclusions:**

YOLOv5: mAP@0.5 of 0.541 on the test set, indicating strong performance YOLOv5 is a single stage detector, it is fast and efficient, with simpler architecture it is better for real time applications like security cameras and autonomous driving

Faster R-CNN: mAP@0.5 of 0.56 on the test, slightly higher than YOLOv5. Faster R-CNN is a two-stage detector, it has better localization accuracy.

Faster R-CNN outperforms YOLOv5 in terms of localization accuracy. However, YOLOv5 with faster inference time, makes it faster to get detection result, which can increase the frames that can handle per second. It can enhance real time detection capabilities.

Faster R-CNN achieved good performance for large objects. It is a better choice to detect cars and large objects. YOLOv5 has a more balanced accuracy across classes of different sizes. It performs better at detecting smaller objects like pedestrians.

**Limitations for Faster R-CNN:**

Training Time: Due to model complexity it takes much longer training time for Faster R-CNN. It takes more than 24 hours to train the model.

Complexity: The version of different libraries has to be a specific version to enable cuda. It takes a long time to set up the environment.

Poor accuracy on small objects with 0.26 AP, due to fixed anchor sizes.

**Improvement Strategies:**

Faster R-CNN:

Implement data augmentation to improve small object detection.

Try anchor box in different sizes in the RPN to better capture small objects.

Use a larger batch size to speed up the training.

YOLOv5:

Explore larger YOLOv5 models (YOLOv5l or YOLOv5x) to increase accuracy.

Comparison with Published Results on the ROAD Dataset

**Published Results:**

The Occluded Object Detection for Autonomous Vehicles Employing YOLOv5, YOLOX and Faster R-CNN research paper (Mostafa, 2022) published results for similar object detection. YOLOv5 achieved mAP@0.5 of 0.777. Faster R-CNN achieved mAP@0.5 0.6883.

YOLOv5: The mAP@0.5 of 0.541 is lower than published results.

Faster R-CNN: The mAP@0.5 of 0.56 lower than published results.

In summary, the performance difference between my ROAD dataset results and Mostafa is due to dataset complexity. ROAD dataset consist of real word videos with different road conditions and complicated urban environments. It is more complex compared to controlled custom dataset used in Mostafa's research paper.

**References:**

Naminas, K. (2025). Object Detection: Key Metrics for Computer Vision Performance. Available at: https://labelyourdata.com/articles/object-detection-metrics#:~:text=Object%20detection%20performance%20is%20measured,comparing%20predictions%20with%20ground%20truth.

Singh, G., Akrigg, S., Maio M. D., Fontana, V. (2023). ROAD: The Road Event Awareness Dataset for Autonomous Driving. Available at: https://www.computer.org/csdl/journal/tp/2023/01/09712346/1AZL0P4dL1e

Ren, S., He K., Girshick, R., Sun J. (2015) Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. Available at: https://arxiv.org/abs/1506.01497

Mostafa, T., Chowdhury, S. J., Rhaman M. K., Alam M. G. R. (2022). Occluded Object Detection for Autonomous Vehicles Employing YOLOv5, YOLOX and Faster R-CNN. Available at: https://www.researchgate.net/publication/365675642_Occluded_Object_Detection_for_Autonomous_Vehicles_Employing_YOLOv5_YOLOX_and_Faster_R-CNN