

**END-TO-END OBJECT DETECTION  
WITH TRANSFORMERS**

**EDDY LAI THIN JUN**

**UNIVERSITI TUNKU ABDUL RAHMAN**

**END-TO-END OBJECT DETECTION  
WITH TRANSFORMERS**

**EDDY LAI THIN JUN**

**A project report submitted in partial fulfillment of the  
requirements for the award of Bachelor of Mechatronics  
Engineering with Honours**

**Lee Kong Chian Faculty of Engineering and Science  
Universiti Tunku Abdul Rahman**

**April 2024**

**DECLARATION**

I hereby declare that this project report is based on my original work except for citations and quotations which have been duly acknowledged. I also declare that it has not been previously and concurrently submitted for any other degree or award at UTAR or other institutions.

Signature :   
\_\_\_\_\_

Name : Eddy Lai Thin Jun  
\_\_\_\_\_

ID No. : 19UEB01182  
\_\_\_\_\_

Date : 20<sup>th</sup> May 2024  
\_\_\_\_\_

**APPROVAL FOR SUBMISSION**

I certify that this project report entitled “**TITLE TO BE THE SAME AS FRONT COVER, CAPITAL LETTER, BOLD**” was prepared by **STUDENT’S NAME** has met the required standard for submission in partial fulfilment of the requirements for the award of Bachelor of Mechatronics Engineering with Honours at Universiti Tunku Abdul Rahman.

Approved by,

Signature :   
\_\_\_\_\_

Supervisor : Ir. Ts. Dr. Tham Mau Luen  
\_\_\_\_\_

Date : 20 May 2024  
\_\_\_\_\_

Signature :   
\_\_\_\_\_

Co-Supervisor : Dr. Kwan Ban Hoe  
\_\_\_\_\_

Date : 20 May 2024  
\_\_\_\_\_

The copyright of this report belongs to the author under the terms of the copyright Act 1987 as qualified by Intellectual Property Policy of Universiti Tunku Abdul Rahman. Due acknowledgement shall always be made of the use of any material contained in, or derived from, this report.

© 2024, Eddy Lai Thin Jun. All right reserved.

## ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my supervisor, Ir. Ts. Dr. Tham Mau Luen. With his invaluable advice and hardware support, I can do the research well and implement the models successfully. I also thank my co-supervisor, Dr. Kwan Ban Hoe for his guidance and support in the study, and also helping make my conference paper be submitted successfully. Lastly, I am also grateful to my fellow friends who have helped me during the training of models. With their contribution, my study can be conducted well and finished faster.

## ABSTRACT

In the past decade, You Only Look Once (YOLO) series has become the most prevalent framework for object detection owing to its superiority in terms of accuracy and speed. However, with the advent of transformer-based architecture, there has been a paradigm shift in developing real-time detector models. This thesis aims to investigate the performance of YOLOv8 and Real-Time DEtection TRansformer (RT-DETR) variants in the context of urban zone aerial object detection tasks. Specifically, a total of five models namely YOLOv8n, YOLOv8s, YOLOv8m, RT-DETR-r18, and RT-DETR-r50 are trained using an expensive graphics processing unit (GPU) and subsequently executed on a central processing unit (CPU), which is more relevant for power-hungry drone applications. Experiment results reveal that RT-DETR-r50 stands out with the highest mean average precision 50-95 (mAP 50-95) of 0.598, whereas YOLOv8n achieves the fastest inference speed of 30.4 frames per second (FPS). Such benefits come at the expense of slow speed (1.7 FPS) and poor accuracy (mAP 50-95 of 0.440), respectively. In this sense, YOLOv8s emerges as the most promising model due to its ability in striving the best tradeoff between accuracy (mAP 50-95 of 0.529) and speed (11.4 FPS).

## TABLE OF CONTENTS

<b>DECLARATION</b>		<b>i</b>
<b>APPROVAL FOR SUBMISSION</b>		<b>ii</b>
<b>ACKNOWLEDGEMENTS</b>		<b>iv</b>
<b>ABSTRACT</b>		<b>v</b>
<b>TABLE OF CONTENTS</b>		<b>vi</b>
<b>LIST OF TABLES</b>		<b>viii</b>
<b>LIST OF FIGURES</b>		<b>ix</b>
<b>LIST OF SYMBOLS / ABBREVIATIONS</b>		<b>ix</b>
<b>LIST OF APPENDICES</b>		<b>xii</b>
<b>CHAPTER</b>		
<b>1</b>	<b>INTRODUCTION</b>	<b>1</b>
	1.1 General Introduction	1
	1.2 Importance of the Study	3
	1.3 Problem Statement	3
	1.4 Aim and Objectives	4
	1.5 Scope of the Study	5
	1.6 Limitation of the Study	5
	1.7 Contribution of the Study	6
	1.8 Outline of the Report	6
<b>2</b>	<b>LITERATURE REVIEW</b>	<b>7</b>
	2.1 Introduction	7
	2.2 Convolutional Neural Network (CNN)	8
	2.3 Overview of Faster R-CNN	9
	2.4 Overview of YOLO	10
	2.4.1 YOLOv5	11
	2.4.2 YOLOv8	12
	2.5 Overview of Transformer	14
	2.5.1 Self-Attention Mechanism in Transformer	15



	2.5.2 Pipeline of Transformer	16
	2.6 Residual Network (ResNet)	19
	2.7 Detection Transformers (DETR)	21
	2.7.1 Pipeline of DETR	21
	2.7.2 Loss Function	23
	2.8 Real-Time Detection Transformers (RT-DETR)	25
	2.9 Summary	26
<b>3</b>	<b>METHODOLOGY AND WORK PLAN</b>	<b>28</b>
	3.1 Introduction	28
	3.2 Work Plan	28
	3.3 Methodology	29
	3.3.1 Similar Studies	30
	3.3.2 Hardware	30
	3.3.3 Software	30
	3.3.4 Datasets	32
	3.3.5 Training Configuration	33
	3.3.6 Testbed	35
	3.4 Unified Pipeline of Real-Time Object Detection	36
	3.5 Gantt Chart	38
	3.6 Summary	38
<b>4</b>	<b>RESULTS AND DISCUSSION</b>	<b>40</b>
	4.1 Introduction	40
	4.2 Evaluation Metrics	40
	4.3 Performance of Models in Evaluation Test	42
	4.4 Performance of Models in Video Processing	44
	4.5 Impact of FPS in Real-Time Object Detection	48
	4.6 Analysis on the Evaluation Results	49
	4.7 Summary	50
<b>5</b>	<b>CONCLUSIONS AND RECOMMENDATIONS</b>	<b>51</b>
	5.1 Conclusion	51
	5.2 Recommendations for future work	51
	<b>REFERENCES</b>	<b>53</b>
	<b>APPENDICES</b>	<b>58</b>

**LIST OF TABLES**

Table 2.1:	YOLOv8 Models Information	13
Table 3.1:	Table of Specification of Training Platform	32
Table 3.2:	Training Models Information	33
Table 3.3:	Configuration of Training Parameters	34
Table 4.1:	Results of Each Models	43
Table 4.2:	mAP(50) of Object Classes in Each Model	44
Table 4.3:	FPS in Each Model	48

**LIST OF FIGURES**

Figure 2.1:	Network Architecture of YOLOv5 (Xu, et al., 2021)	12
Figure 2.2:	Basic Architecture of YOLOv8 (Agarwal, et al., 2023)	14
Figure 2.3:	Architecture of Transformer (Vaswani, et al., 2023)	17
Figure 2.4:	Architecture of ImageNet (He, et al. 2016)	21
Figure 2.5:	Architecture of DETR (Carison, et al., 2020)	21
Figure 2.6:	Architecture of DETR's Transformer (Carison, et al., 2020)	23
Figure 2.7:	Pipeline of DETR (Carison, et al. 2020)	23
Figure 2.8:	Basic Architecture of RT-DETR (Lv, et al. 2023)	26
Figure 3.1:	Methodology of the Study	29
Figure 3.2:	Flowchart of Models' Training	35
Figure 3.4:	Gantt Chart Phase 1	38
Figure 3.5:	Gantt Chart Phase 2	38
Figure 4.1:	Chart of Results of Each Models	43
Figure 4.2:	Chart of mAP(50) of Object Classes in Each Model	44
Figure 4.3:	YOLOv8n Aerial Object Detection in Video	45
Figure 4.4:	YOLOv8s Aerial Object Detection in Video	45
Figure 4.5:	YOLOv8m Aerial Object Detection in Video	46
Figure 4.6:	RT-DETR-r18 Aerial Object Detection in Video	46
Figure 4.7:	RT-DETR-r50 Aerial Object Detection in Video	47
Figure 4.8:	Chart of FPS in Each Model	48

## LIST OF SYMBOLS / ABBREVIATIONS

$b_i$	ground truth bounding box vector
$\hat{b}_i$	predicted bounding box vector
$c_i$	target class label vector
$\hat{c}_i$	predicted class vector
$\mathcal{L}_{match}$	pairwise matching cost
$\hat{p}$	probability
$\mathcal{Y}_i$	ground truth index
$\hat{\mathcal{Y}}_{\sigma(i)}$	prediction with index
$\sigma(i)$	index within a particular permutation of N elements
Adam	Adaptive Moment Estimation optimizer
AdamW	Adam with Weight Decay Regularization
AI	Artificial Intelligence
AP	Average Precision
BCE	Binary Cross Entropy
ChatGPT	Chat Generative Pre-training Transformer
CIoU	Complete Intersection over Union
CNN	Convolutional Neural Network
COCO	Common Objects in Context
CPU	Central Processing Unit
CSPNet	Cross Stage Partial Network
CUDA	Compute Unified Device Architecture
cuDNN	NVIDIA CUDA Deep Neural Network library
DETR	Detection Transformer
eGPU	External Graphics Processing Unit
FAIR	Facebook's AI Research lab
Faster R-CNN	Faster Region-based Convolutional Neural Networks
FFN	Feed-forward Network
FN	False Negative
FP	False Positive
FPS	Frames Per Second

GPU	Graphics Processing Unit
GB	GigaByte
HSV	Hue Saturation Value
IoU	Intersection over Union
mAP	Mean Average Precision
OpenCV	Open Source Computer Vision Library
P	Precision
PANet	Path Aggregation Network
R	Recall Value
RAM	Random Access Memory
R-CNN	Region-based Convolutional Neural Networks
ReLU	Rectified Linear Unit
ResNet	Residual Neural Network
RoI	Region of Interest
RPN	Region Proposal Network
RT-DETR	Real-Time Detection Transformer
RT-DETR-r18	Real-Time Detection Transformer with Resnet-18
RT-DETR-r50	Real-Time Detection Transformer with Resnet-50
SGD	Stochastic gradient descent
SiLU	Sigmoid Linear Unit
SPP	Spatial Pyramid Pooling
SSD	Single Shot Detector
TP	True Positive
VGG	Visual Geometry Group
YOLO	You Only Look Once
YOLOv5	You Only Look Once version 5
YOLOv7	You Only Look Once version 7
YOLOv8	You Only Look Once version 8
YOLOv9	You Only Look Once version 9

**LIST OF APPENDICES**

Appendix A: Figures	58
Appendix B: Graphs	62

## CHAPTER 1

### INTRODUCTION

#### 1.1 General Introduction

In the past few years, with the advancement of technology, computer vision has become more and more important. It can be used in many fields, such as transportation, communication, medicine etc. One of the elements is object detection, which is the identification and localization of objects of interest in images or video streams. Traditionally, the traditional object detection approach is using machine learning to detect objects. However, it needs a human to intervene and gives less accuracy. Then, deep learning is coming. Deep learning requires more data to train but has a lot of improvement in accuracy compared to machine learning. One of the famous examples is convolutional neural networks (CNN). It relies on its own strength, demonstrating impressive results in various applications and becoming the main approach of object detection now. In particular, the CNN-based YOLO (You Only Look Once) appearance subverts the previous object detection models. It has a faster detection speed and higher accuracy, so it has been applied to many object detection fields, implemented in many real-time object detection applications and occupied for a long time (Jason, et al., 2022).

Nowadays, the drone's technology has also significantly advanced. They are no longer just capable of flying, now most of the drones have their own camera so that they can also capture photos and videos. Due to this capability, they can effectively manage a city, especially in terms of traffic management, urban planning and law enforcement. Therefore, aerial detection of objects on land is becoming increasingly important.

In order to detect objects, there are many powerful object detection models now, such as YOLO and DETR (Detection Transformer). They both have high accuracy object detection performance and have thus attracted great attention. Among all the YOLO models, YOLOv8 represents the latest advancement in the YOLO series, known for its good real-time object detection performance with a high accuracy result. Because of its advanced architecture and algorithms, accurate and efficient object detection makes it widely used in

industries like robotics, video surveillance, and autonomous driving. Hence, this makes it positioned nearly at the pinnacle of real-time object detection models.

Conversely, DETR, a new member to the object detection landscape, was introduced in a paper “End-to-End Object Detection with Transformers” in 2020 (Carison, et al., 2020). The emergence of DETR has surprised researchers in the field of computer vision and given a new way for implementing object detection applications. Unlike traditional CNN object detection methods that use anchor generation or non-maximum suppressors, DETR uses a fully attention-based mechanism to predict object instances directly from the input image (Carison, et al., 2020). Originally, Transformer was only used for natural language processing tasks, but it was later found that it could be used for object detection, and it seems to have excellent potential behind them.

However, DETR lacks the speed required for real-time inference even with its innovative approach. To overcome this limitation, Baidu introduced a new DETR model, RT-DETR (Real-Time Detection Transformer), by a paper “DETRs Beat YOLOs on Real-time Object Detection” in 2023. It is designed for the implementation of real-time end-to-end object detection based on the architecture of Transformer. In order to do so, it uses an IoU-aware query and an efficient hybrid encoder mechanism to support its real-time object detection performance. The paper shows the supremacy of RT-DETR, by comparing RT-DETR against real-time end-to-end object detectors like YOLO, PPYOLOE, and Efficient-DETR. Based on the paper, RT-DETR-L achieves an average precision (AP) of 53.0% and operates at 114 frames per second (FPS), while RT-DETR-X attains 54.8% AP and 74 FPS, showing a better performance than YOLO.

Hence, in this study, RT-DETR is utilized for comparison with YOLO in object detection tasks using the Urban Zone Aerial Object Detection Dataset (UZAODD). In this dataset, there are four primary classes of objects, which are persons, small vehicles, medium vehicles, and large vehicles. All these classes commonly appeared in urban zones. By evaluating the performance of RT-DETR and YOLOv8 on this standard dataset, their strengths and limitations can be identified so that the best object detection method can be determined.



## **1.2 Importance of the Study**

Object detection can be said as a cornerstone of computer vision applications. It can be used in many applications, such as facial recognition, industrial quality checking, people counting etc. Thus, as the domain of computer vision progresses, it is very important to understand the current limitations of object detection approaches, such as the YOLOv8 model which is based on the CNN architecture.

Nowadays, many applications need a more accurate and faster response object detection model, efficient and accurate object detection is a needed for enabling machines to interact intelligently with their environment and make informed decisions. If a more efficient and accurate object detection model is found, it will be a huge improvement in many sectors, such as transportation, communication, and medicine, when object detection is applied to those sectors. Thus, it is very important to investigate and improve current object detection technology. Currently, the mature object detection technology is the CNN-based YOLOv8 model (Hussain, 2023). It is fast in the response speed and accurate in the result of object detection. A new object detection approach, DETR, has been introduced, which is based on the Transformer architecture. It is different from the traditional CNN object detection approach (Carison, et al., 2020). Now, the newest of the Transformer-based architecture object detection models is RT-DETR (Lv, et al., 2023). Hence, a comparative analysis study has been carried out between the traditional CNN approach object detection model, YOLOv8 model and the Transformer-based architecture object detection model, RT-DETR. Besides the limitations, it is also important to know the difference between new object detection technology and mature object detection technology, so that the object detection technology can be improved.

## **1.3 Problem Statement**

Object detection is getting important in these few years. However, current object detection technology is still not perfect, even using the mature object detection technology such as the YOLOv8 model. Although the YOLOv8 model has demonstrated considerable success in object detection, it is not without limitations.

Nowadays, object detection is facing some challenges such as accuracy, speed, and complexity. Although the CNN-based models like YOLOv8 are the popular model for doing real-time object detection, they are still not perfect in accuracy and inference speed. Moreover, CNN-based models also heavily rely on anchor boxes and complex post-processing steps, which can be cumbersome and less intuitive. Additionally, the architecture of CNN is complex and makes it challenging to scale it effectively to different sizes of datasets.

Currently, a new object detection technology, the RT-DETR model based on Transformer architecture, may beat the YOLOv8 model in object detection in the form of accuracy and speed. Thus, a study is carried out to investigate the difference between both models in object detection.

#### **1.4 Aim and Objectives**

This study aims to implement two object detection models, the YOLOv8 model based on CNN architecture and the RT-DETR model based on Transformer architecture. This project is focused on analyzing the results between them in real-time object detection performance, whether the Transformer-based architectures with their self-attention mechanisms can overcome the current object detection's limitations and offer superior performance in object detection tasks. The objectives of the study are as follows:

- (i) To study Transformer-based architectures, including both the encoder and decoder components.
- (ii) To implement an end-to-end object detection model using Transformer architecture.
- (iii) To evaluate and compare the performance of Transformer-based object detection models with traditional CNN-based approach models on a standard dataset.

All the objectives in this study were achieved successfully. The Transformer architecture, including both the encoder and decoder components, were well known before implementing the object detection models. The RT-DETR models based on Transformer architecture, and the YOLOv8 model based on CNN architecture, were implemented successfully on a Urban Zone

Aerial Object Detection Dataset. Lastly, a comparative analysis of RT-DETR and YOLOv8 was successfully conducted which compared their performance.

### **1.5 Scope of the Study**

This project covers the implementation of object detection model by using RT-DETR based on Transformer architecture and YOLOv8 based on CNN architecture. First, this study discusses the details of current object detection situation. YOLO is the most mature and popular model for real-time object detection now. Then, a new object detection model, RT-DETR was released which is based on Transformer architecture. Then, the details of Transformer is discussed including the encoder and decoder. The details of DETR and RT-DETR are also covered in the literature review. Beside the Transformer, this study also covers the overview of CNN, YOLO, ResNet, etc.

There are variant weights of RT-DETR models have been released, RT-DETR-r18 and RT-DETR-r50 are choosing as the training models. On the YOLO side, YOLOv8n, YOLOv8s, and YOLOv8m are selecting as the compared models as they are the lightest weight models. The dataset is an Urban Zone Aerial Object Detection Dataset which is downloaded from Kaggle (Sganderla, 2021). After all the models done training, some evaluation metrics are used to evaluate the models' performance such as FPS, mAP, and recall rate.

### **1.6 Limitation of the Study**

There are some limitations to this study. First, the limitation of GPU memory. The GPU of the training platform has only 10GB. It is not sufficient to train high batch-size training in heavier models like YOLOv8m. This means the batch size of training YOLOv8m cannot be consistent with YOLOv8n and YOLOv8s. Therefore, the training rate cannot be the optimum unless a huge memory of GPU is provided for training.

Besides that, the RT-DETR and YOLOv8 models may not be the newest Transformer-based architectures and CNN-based architecture object detection models as they are evolving rapidly in deep learning research. Furthermore, the evaluation might be limited due to time and resource constraints. The training datasets may not be good enough to train the models.

Hence, a further exploration of various model configurations is left for future research.

### **1.7 Contribution of the Study**

The main contribution of this project is the development of two Urban Zone Aerial Object Detection models that are able to detect person, small vehicle, medium vehicle, and large vehicle in an urban zone. One of the models is trained by using RT-DETR and another one is trained by using YOLOv8. Another contribution lies in the comparative analysis of the Transformer-based RT-DETR model and the traditional CNN-based YOLOv8 model for end-to-end object detection. Through the evaluation of performance metrics and a detailed discussion on the strengths and weaknesses inherent in each approach, a thorough comparison of the RT-DETR and YOLOv8 models becomes feasible. This facilitates the determination of which model excels in Urban Zone Aerial Object Detection.

### **1.8 Outline of the Report**

The report first covers the introduction of the report, which includes the problem statement, aim and objectives, contribution, limitation, and scope of study. In Chapter 2, a literature review of a particular topic which discusses the details of current object detection, CNN, YOLOv8, Transformer, DETR and the RT-DETR. Then, in chapter 3, the methodology and approach of the study are reviewed. In this chapter, the method of implementation of the models, the parameters details of the models, details of the training datasets and the work plan are introduced in detail. The next chapter is followed by results and discussion. It discusses the evaluation results and analyzes the performance between the models. Lastly, the conclusion and recommendations of the study are discussed in the last chapter. In this chapter, the summary of the whole study and recommendations for future work are sorted out. The references are listed after the chapter.

## CHAPTER 2

### LITERATURE REVIEW

#### 2.1 Introduction

Nowadays, object detection has become more important in computer vision. This is because more and more people see that it has achieved many applications, such as object tracking, image captioning, instance segmentation, and many others. Now, pedestrian, animal, vehicle, and face detection are popular projects in the object detection field. Besides that, object detection is the main mission in computer vision, which requires locating and identifying objects in videos or images. Its main goal is not just object recognition, but also the precise object's location via the bounding boxes. Therefore, object detection is significantly more challenging than image classification, in addition to accurately classifying the object but also simultaneously predicting its location on the image (Arkin, et al., 2022).

Currently, deep learning has changed the object detection past primary models and algorithms, bringing the computer vision to a new era. Among the contemporary state-of-the-art models, prominent deep learning architectures have surfaced, yielding remarkable outcomes in object detection endeavors. For example, YOLO (You Only Look Once), SSD (Single Shot Detector), and R-CNN (Region-based Convolutional Neural Networks). These models are becoming more popular because of their effectiveness in detecting objects (Jason, et al., 2022).

Before entering the era of deep learning, object detection basically relied on handcrafted features and traditional machine learning techniques. The integration of deep learning algorithms, like CNN, has transformed the field of object detection. One- and two-stage object detection algorithms, exemplified by models like YOLO, SSD, and R-CNN have become standard approaches in the field now (Jason, et al., 2022).

Before, almost all object detection algorithms used CNN as their backbone until the paper for Transformers was published. Transformer in AI is a deep learning architecture. It has first prominence in the natural language processing (NLP) field, getting some good achievements in the NLP field, such

as the introduction of ChatGPT. Recently, Transformer-based object detection methods have shown some good potential, indicating that they may replace traditional CNN object detection methods (Arkin, et al., 2022). For example, the paper 'End-to-End Object Detection with Transformers' introduces a new target detection model, DETR, based on the Transformer architecture. The key idea behind DETR is to approach object detection as a direct ensemble prediction problem, where the model predicts bounding boxes and their corresponding object classes simultaneously in an end-to-end manner (Carison, et al., 2020). Now, many new DETR object detection models are developed, like RT-DETR, Efficient DETR, Deformable DETR, etc.

## **2.2 Convolutional Neural Network (CNN)**

Convolutional Neural Network (CNN), is one of the deep learning algorithms. It is found that it was excellently tailored for tasks in recognition and image processing. Thus, it is used in many sectors about image analysis, such as facial recognition, image classification and object detection (Alzibaidi, et al., 2023). CNN consist of several layers, which are fully connected layers, pooling layers, and convolutional layers. All these layers are used to help in the extraction of image features, empower CNN to recognize objects or characteristics within images without being reliant on their specific positions, thereby effectively diminishing the intricacy of the network (Patel and Patel, 2020).

The core concept of CNN lies in convolution and filter utilization. Convolutional layer filters are applied to the input images, then these filters will move across the images to generate feature maps that capture specific attributes of the images. All these filters worked together simultaneously to detect diverse visual features in order to get the complete attributes' information of the input images. Besides that, pooling techniques are used to decrease the dimensions of feature maps but retain the critical features of the input images. Not only that, some pooling like average and max pooling are also used to achieve spatial down-sampling, improving computational efficiency and avoid the over-fitting problem. Furthermore, activation functions introduce non-linearity into the system, it allows the CNN to handle some complex feature connections. Popular activation functions like tanh, sigmoid, and ReLU have different benefits,

limitations, and characteristics, affecting the network's learning process (Zhao, et al., 2019).

Over the years, CNN architecture has been improved a lot. Prominent models such as LeNet-5, AlexNet, VGGNet, and ResNet, as well as variations like MobileNet, DenseNet, and InceptionNet have played significant roles in the field of computer vision (Zhao, et al., 2019). All these architectures are designed to solve specific problems of computer vision tasks. Additionally, CNN has also found that it can be used in many applications besides image classification. They do well in object detection tasks, where they can classify and give the precise location of the objects in an image. Besides that, semantic segmentation allows CNN to assign class labels to each pixel, facilitating detailed scene analysis (Galvez, et al., 2018).

As a short summary, CNN is very important to the world now as it is applied to many applications. These include healthcare, agriculture, retail, security, entertainment etc.

### **2.3 Overview of Faster R-CNN**

Faster R-CNN is a object detection model that improved from its predecessor, RCNN (Region-based Convolutional Neural Network). Proposed by Shaoqing Ren, and his team in 2015, Faster R-CNN aims to achieve real-time object detection performance while maintaining high accuracy.

Faster R-CNN's primary breakthrough centers on the incorporation of the Region Proposal Network (RPN) directly into the object detection framework. Unlike RCNN which relied on external region proposal methods like selective search, Faster R-CNN seamlessly generates region proposals as part of the network, eliminating the need for time-consuming external processes (Ren, He, Girshick and Sun, 2016).

The Faster R-CNN architecture comprises two primary modules: the Region of Interest (RoI) Pooling and the Region Proposal Network (RPN). The RPN is a streamlined neural network that utilizes an identical backbone as the object detection network, such as ResNet. It operates on the feature maps extracted from this backbone and adeptly produces region proposals, which denote potential bounding boxes expected to encompass objects of interest. These proposals are generated based on anchor boxes, predefined bounding box

shapes with different aspect ratios and scales. The RPN predicts the offset and size adjustments for each anchor box to generate the final region proposals (Ren, He, Girshick and Sun, 2016).

Once the region proposals are generated by the RPN, they are passed through RoI pooling, a process that extracts fixed-size feature representations from the proposed regions. These RoI features are then fed into fully connected layers for object bounding box and regression classification. The classification head uses softmax to predict the probability of each proposal belonging to specific object classes, while the bounding box regression head refines the locations of the proposals to improve localization accuracy (Ren, He, Girshick and Sun, 2016).

Faster R-CNN's end-to-end training allows the network to learn region proposals specific to the task, which enhances its accuracy and efficiency. By seamlessly integrating the region, proposal steps into the network. As a result, Faster R-CNN attains real-time object detection capabilities without compromising its high level of accuracy (Ren, He, Girshick and Sun, 2016).

## **2.4 Overview of YOLO**

“You Only Look Once” (YOLO) is a popular object detection model in the computer vision field. It was first introduced by Joseph Redmon and his team with the paper "You Only Look Once: Unified, Real-Time Object Detection" in 2015 (Redmon, et al., 2016). Currently, YOLO models can be said as the most popular object detection models as it is light weight but high accuracy in performance. One of its key advantages is that its object detection speed is fast, making it accessible for predicting the objects in real-time (Gašparović, et al., 2023).

First, YOLO will divide the image into a grid of  $N \times N$  cells. In the each grid, the YOLO will calculate the class probabilities and bounding box parameters which include x, y coordinates, width and height. Finally, the class prediction and the confidence score for the bounding box are amalgamated to generate a conclusive score, signifying the likelihood that the bounding box encompasses a specific object category (Redmon, et al., 2016).



### 2.4.1 YOLOv5

YOLOv5 was the most famous version of YOLO until 2023. It was introduced in 2020, and represents the fifth generation of the YOLO series, which has been refined and improved to achieve good accuracy in real-time object detection tasks. Because of its highly efficient and good performance, it quickly gained popularity in the computer vision and deep learning communities due to its exceptional speed and accuracy (Jocher, et al., 2022).

There are 3 main components in the YOLOv5 architecture, which are backbone, neck, and a head. The backbone of YOLOv5 is CSP-Darknet53. This backbone is essentially the convolutional network Darknet53 applied with a Cross Stage Partial (CSP) network strategy. The next component is the Neck, which includes Spatial Pyramid Pooling (SPP) and Path Aggregation Network (PANet). In YOLOv5, a variant of Spatial Pyramid Pooling is used called SPPF, and the Path Aggregation Network is modified to incorporate the CSPNet strategy. Finally, the head component of YOLOv5 is responsible for the concluding operations. It deploys anchor boxes on the feature maps and generates the ultimate output, encompassing class predictions, bounding boxes, and objectness scores. (Xu et al., 2021).

There are 5 models on the YOLOv5 with different sizes, which are n, s, m, l, and x, followed by increase of parameters. Although the parameters are different in each model, all the components remain the same in these 5 models. Also, YOLOv5 makes use of the SiLU and Sigmoid activation functions, and applies loss functions like Binary Cross Entropy (BCE) and Complete Intersection over Union (CIoU) to compute different aspects of the model's outputs (Al-Smadi et al., 2023).

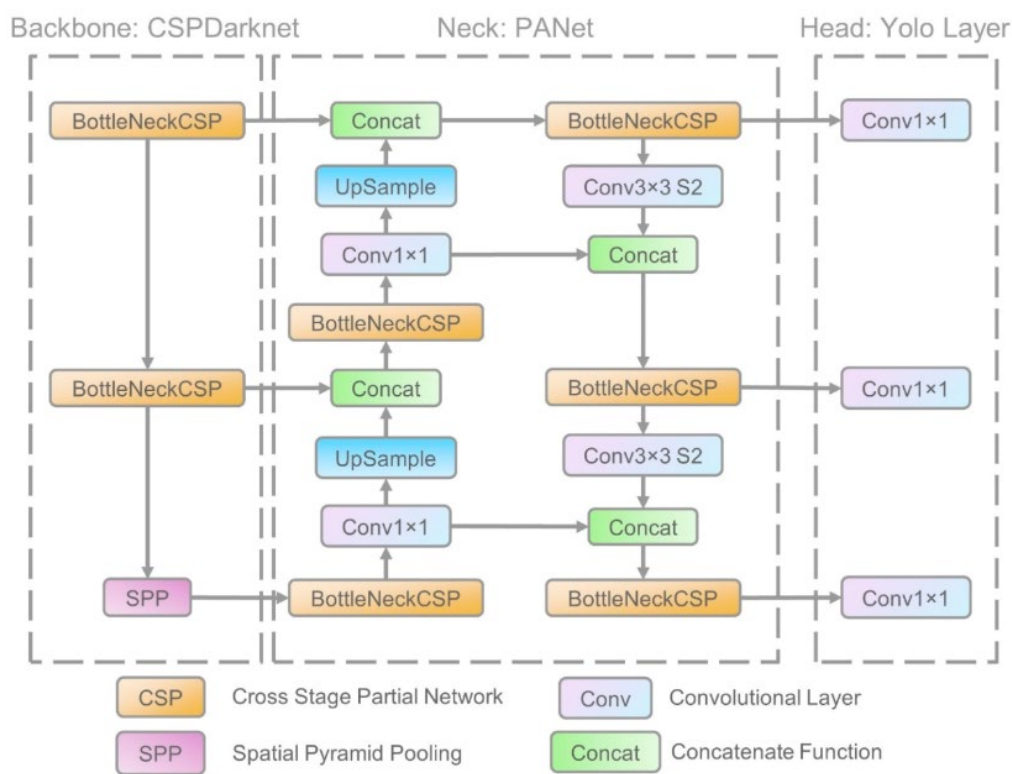


Figure 2.1: Network Architecture of YOLOv5 (Xu, et al., 2021)

## 2.4.2 YOLOv8

YOLOv8, is an evolution of the YOLO object detection model, epitomizes a cutting-edge CNN-based architecture for object detection. Although it does not have an official paper, it is the latest iteration and integrates advancements in architecture design and training techniques in order to achieve improved accuracy and speed compared to its predecessors. YOLOv8 uses a modified Darknet architecture as its backbone, and a few enhancements such as feature pyramid networks (FPNs) and advanced data augmentation.

Developed by Jocher et al., YOLOv8 builds upon the foundation of the YOLOv5 architecture with a series of enhancements and extensions introduced by Ultralytics. These improvements mainly focus on the model scaling and architecture adjustments detailed in the code and documentation are available in the Ultralytics YOLOv8 repository. Furthermore, YOLOv8 also has several improvements to other YOLO, such as enhanced feature representation, better handling of small objects, and increased training efficiency. This makes the YOLOv8 become the famous choice when choosing a real-time object detection model.

There are two main components in the YOLOv8 architecture, which are a backbone and a head. The backbone of YOLOv8 is the C2f module, inspired by the ELAN module. Besides that, a variant of Spatial Pyramid Pooling is used in the backbone called SPPF. It is used to extract informative features from images at varying scales. Finally, the head component of YOLOv8 is responsible for the concluding operations. It deploys anchor boxes on the feature maps and generates the ultimate output, encompassing class predictions, bounding boxes, and objectness scores.

There are 5 models in the YOLOv8 with different sizes, which are n, s, m, l, and x, followed by an increase in parameters. Although the parameters are different in each model, all the components remain the same in those 5 models. Below shows the parameters of each YOLOv8 model:

Table 2.1: YOLOv8 Models Information

<b>Model</b>	<b>FLOPs (B)</b>	<b>Params (M)</b>	<b>mAP<sup>val</sup> (50-95) on COCO val2017</b>
YOLOv8n	8.7	3.2	37.3
YOLOv8s	28.6	11.2	44.9
YOLOv8m	78.9	25.9	50.2
YOLOv8s	28.6	11.2	52.9
YOLOv8m	78.9	25.9	53.9

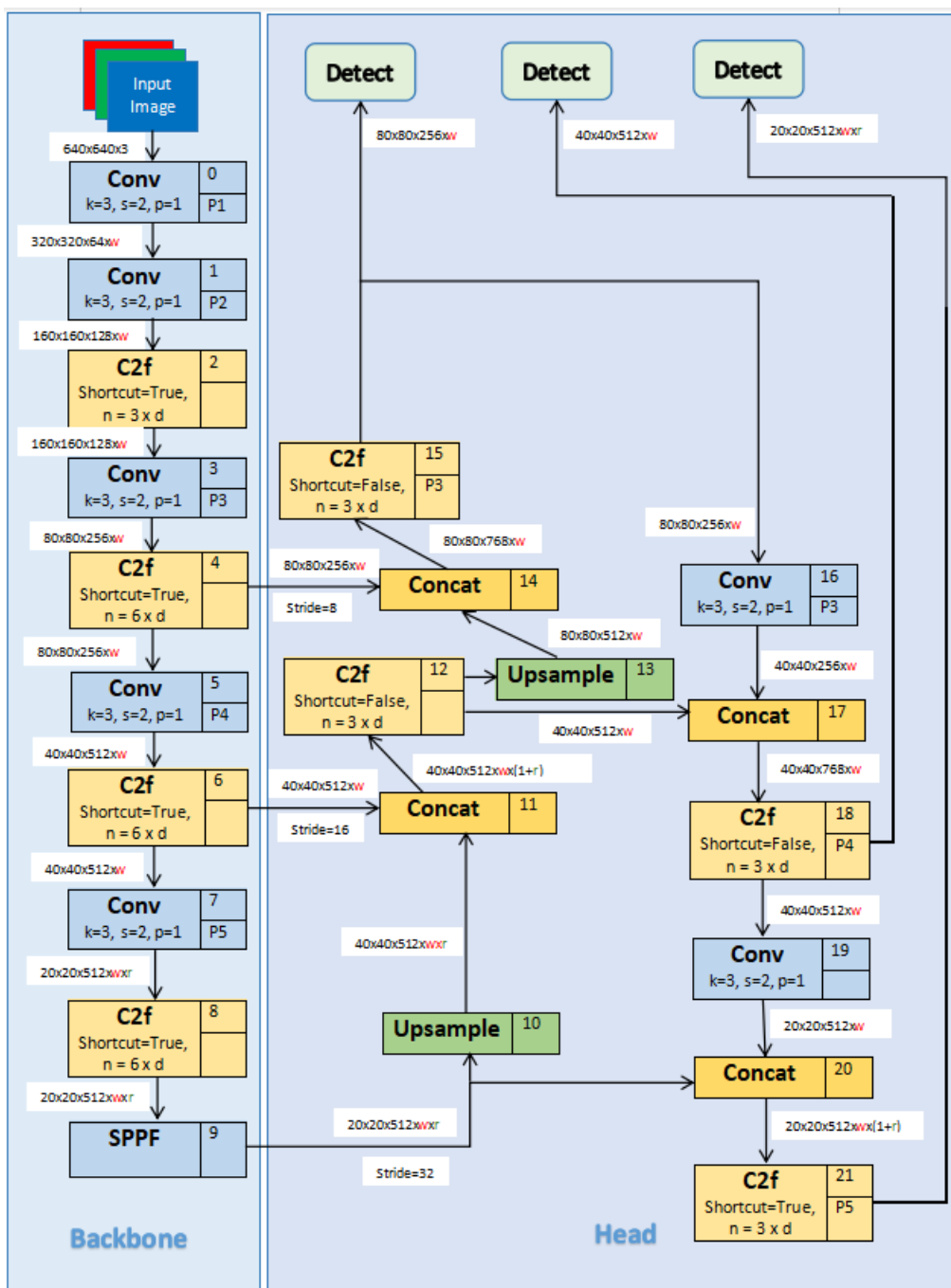


Figure 2.2: Basic Architecture of YOLOv8 (Agarwal, et al., 2023)

### 2.5 Overview of Transformer

In 2023, the Transformer could be said to be very popular. Maybe some people haven't heard of it, but everyone definitely knows about ChatGPT. In fact, the full name of ChatGPT is called Chat Generative Pre-trained Transformer, which is the AI tool that runs on a Transformer architecture (Liu, et al., 2023).

Transformer is a machine learning model architecture that was first introduced in the paper ‘Attention Is All You Need’ by Vaswani and his partners

in 2017. There are two main components in the Transformer, which are encoder and decoder. Unlike RNNs, the Transformer's encoder and decoder use the attention system to process the whole input at once and just focus on the interest parts of the input. This makes the training time of Transformer reduced much more than the RNNs (Vaswani, et al. 2023).

### **2.5.1 Self-Attention Mechanism in Transformer**

Attention can be said as a key to the Transformer. Without it, the Transformer would not be formed today. Attention layers are located in the encoder and decoder parts. Before reaching the attention layer, the input text is first labeled with a representative embedding. From each embedding, the encoder uses the attention to generate a representative encoding, while the decoder does the opposite, outputting the text (Vaswani, et al. 2023).

There are four stages in the attention system. In the first stage, the embedded words in the sentence act as input and pass through the attention layer. Then, it will produce 3 output vectors for each word, which are Queries, Keys, and Values. Thus, there are N Queries, Keys, and Values for N words in the sentence (Unzueta, 2022). Next, the attention layer calculates the score of each word in the second stage. In order to calculate the score of all other words related to each word, the query vector of the word needs to be multiplied by the key vector to get the dot product. Then, the dot product is divided for the scaling purpose (Vaswani, et al., 2023).

In the third stage, the dot product is applied a softmax function to make their value between 0 and 1. After applying the softmax function, Transformer focused on higher score embedding instead of lower score. Finally, the Value vectors are multiplied by them to get the attention vector of each word. Because of the softmax function, the higher scores are still higher, and the model is more interested in them as the model defines them as more important words. Equation (2.1) is as follows:

$$Attention(Q, K, V) = \text{soft max}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (2.1)$$

where

Q = Query vector

K = Key vector

V = Value vector

$d_k$  = dimension of key vector

Note that each result vector depends only on the query for that word, and also on the keys and values of all words in the sentence. This is why attention is powerful in sequential tasks (Vaswani, et al., 2023).

Furthermore, the attention system above is just a single-head attention, there is a multi-head attention layer in the Transformer. The multi-head attention splits more Value, Key, and Query vectors into six groups. Subsequently, these six groups undergo identical self-attention procedures, with each procedure referred to as a "head." Each head generates its own attention vector, which is later combined into a unified vector before passing through a concluding linear layer (Vaswani, et al., 2023).

### 2.5.2 Pipeline of Transformer

There are two main components in Transformer, an encoder and a decoder. In the encoder, the model takes the sentence and vectorizes it, and then transforms it using the attention mechanism. Besides that, the decoder does the opposite by converting vectors to sentences. Here is the Transformer's architecture:

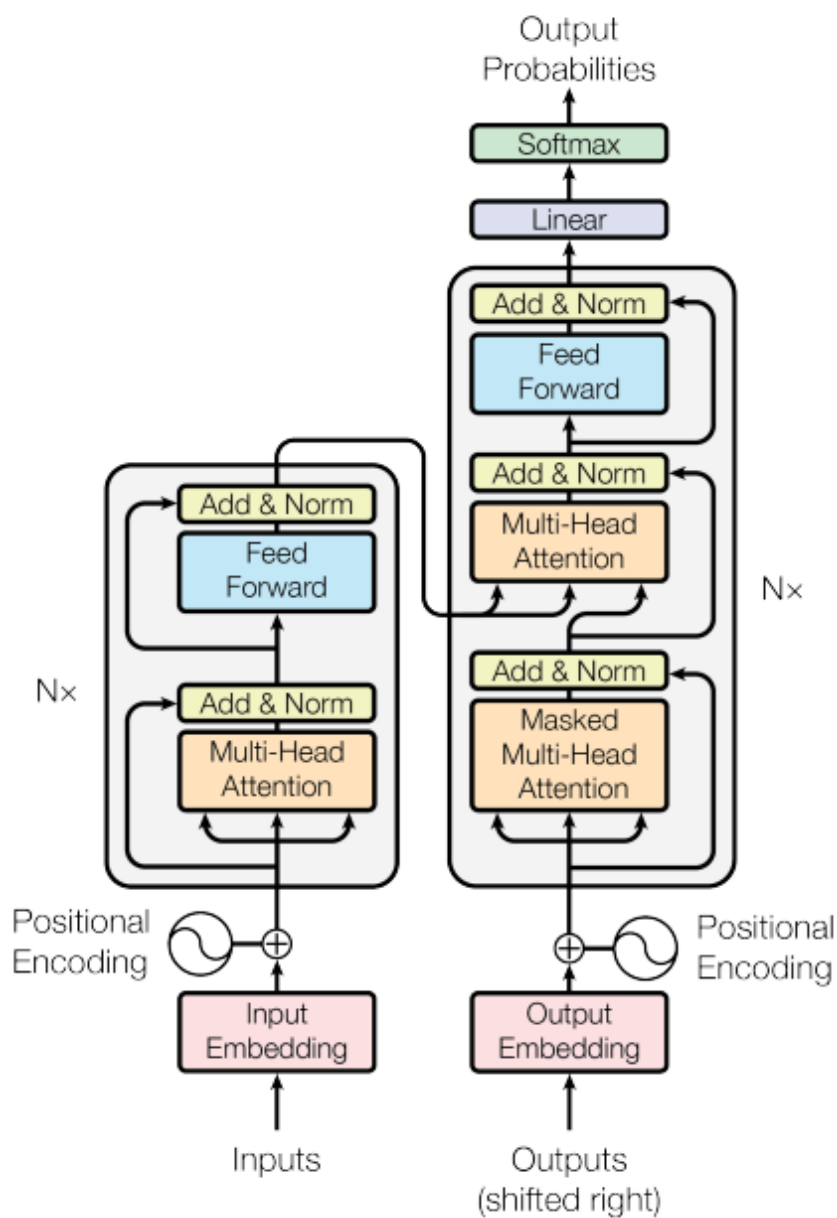


Figure 2.3: Architecture of Transformer (Vaswani, et al., 2023)

At the beginning, the input which is the sentence is sent to the input embedding to transform the input word into an input embedding. Since the Transformer lacks inherent sequential order like RNNs, the concept of positional encoding is introduced to impart information regarding the specific position of each word within the input sequence. It allows model to understand the order of the input data (Gering, et al., 2017). Hence, the Transformer injects a positional encoding into the word embeddings to determine the position of each word in the sentences (Vaswani, et al., 2023).

Then, the input embedding is sent to the multi-head attention in the encoder. With this attention, the encoder employs attention to associate each word with all other words and compute a score for each of them. After multi-head attention, an attention vector is generated and combined with the original input. This step is referred to as a residual connection. The output of it is then subject to layer normalization, and the output is directed into a point-wise feed-forward network for additional refinement. This network has two linear layers separated by a ReLU activation function. Next, the input of the point-wise feed-forward network was also added to its output and further normalized. Residual connections help the gradients to move directly through the network, thus aiding network training. Layer normalization is used to stabilize the network, thus consistently yielding the necessary training time, and the attention output is further processed using a point-wise feed-forward network, potentially giving it a richer representation (Vaswani, et al., 2023).

After the encoder, the output embedding of the encoder, which is attention embedding, is sent to the decoder. However, it is sent to the second stage attention layer instead of the first stage. In the decoder, there are point-wise feed-forward network, three layer normalization, and two multi-head attention layers that integrates residual connection after each sub layer. These sub-layers exhibit behavior similar to those found in encoders, except for the multi-head attention layers, which diverge in function. Since the decoder is autoregressive, it takes its previous outputs as inputs, and also the attention embedding from the encoder (Vaswani, et al., 2023).

At the onset of the decoder phase, an input undergoes processing through both an embedding layer and a positional encoding layer to acquire positional embeddings. These embeddings are subsequently directed into the initial multi-head attention layer, which calculates attention scores for the input provided to the decoder. This multi-head attention layer operates with a slight variation compared to its counterpart in the encoder. Here, a masking technique is applied to prevent the decoder from accessing information about future tokens. In this masking process, a matrix serves as a mask, aligning in size with the attention scores and filled with values of 0 and negative infinity. When the attention scores are combined with this mask, all the future tokens are assigned negative infinity values. Thus, after the softmax function, all the negative



infinities become zero, the decoder will ignore the future tokens as their scores are zero (Vaswani, et al., 2023).

In the second stage, the multi-head attention layer, the attention embedding from the encoder is passed to here, but just the Query vectors and Key vectors. The remaining Value vectors come from the first multi-head attention layer. This second stage multi-head attention layer is used to decide which encoder input is needed to put focus on. After that, the output embedding from this attention layer is sent to the point-wise feed-forward network (Vaswani, et al., 2023).

Finally, the output of the final feed-forward layer is passed to a final linear layer for classification purpose. The output of linear layers goes through a softmax function to get a probability score between 0 and 1. This output of the softmax function is sent back to the beginning of the decoder and acts as the input of the decoder until an ‘End’ token is predicted (Vaswani, et al., 2023).

## **2.6 Residual Network (ResNet)**

Nowadays, the Residual Network is getting popular, such as the ResNet-50 model. ResNet is a type of neural network that was developed by Kaiming He and his team at Microsoft Research in 2015. ResNet was introduced to tackle a critical issue that had been plaguing the training of very deep neural networks: the vanishing gradient problem.

Deep neural networks consist of multiple layers of neurons, and they are exceptionally effective at learning hierarchical representations of data. However, as networks get deeper, they become more challenging to train. One major issue that arises in deep networks is the vanishing gradient problem. During training, the gradients which are derivatives of the loss with respect to the network parameters are computed and used to update the model's weights. In deep networks, these gradients can become tiny as they are propagated backward through the layers. This leads to slow convergence, and in some cases, training can stall completely (He, et al. 2016).

Then, ResNet introduced the concept of residual blocks. Instead of endeavoring to directly learn the desired underlying mapping, residual networks learn residual mappings. These residual mappings denote the disparity between the sought-after output and the current prediction. By learning these residuals,

ResNet aims to make it easier for the network to represent identity mappings where the input matches the output. A residual block consists of two main paths: the shortcut path and the identity path which is also known as the skip connection. Let's break down how a residual block works (He, et al. 2016).

A typical convolutional layer followed by two Rectified Linear Unit (ReLU) activation functions and two batch normalization layers. Instead of directly passing the output of this convolutional layer to the next layer, ResNet introduces a "skip connection" or "shortcut." This connection directly adds the input of the block to the output. Mathematically, it can be represented as:

$$F(x) = H(x) + x \quad (2.2)$$

where

$F(x)$  = output of residual block

$H(x)$  = output of the convolutional layers

$x$  = input to the block.

This architecture allows gradients to flow freely during back propagation because, in the worst case, if the convolutional layers fail to learn anything useful, the skip connection will pass the identity mapping through, ensuring that the gradient doesn't vanish (He, et al. 2016).

Besides that, ResNet architectures come at different depths, typically labeled with numbers like ResNet-18, ResNet-34, ResNet-50, etc. For instance, ResNet-18 has 18 layers, while ResNet-50 has 50 layers. The key difference between ResNet-50 and ResNet-101 is the network's depth and complexity. ResNet-101 has a deeper architecture with more layers and more parameters. Although the increased depth can capture more complex features, the demands on computing resources and data also increase. While ResNet-50 is already known for its exceptional performance and computational efficiency, ResNet-101 is typically reserved for scenarios where very high accuracy is required, and there is access to sufficient datasets and computational resources. In practical, ResNet-50 is a more popular choice due to its balance between performance and efficiency (He, et al. 2016).

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		$1.8 \times 10^9$	$3.6 \times 10^9$	$3.8 \times 10^9$	$7.6 \times 10^9$	$11.3 \times 10^9$

Figure 2.4: Architecture of ImageNet (He, et al. 2016)

## 2.7 Detection Transformers (DETR)

DETR stands for "DEtection TRansformer," and it is an end to end object detection model based on the Transformer architecture which was introduced in the paper "End-to-End Object Detection with Transformers" by Nicolas Carison, et al., published in 2020.

The key idea behind DETR is to approach object detection as a direct set prediction problem, where the model predicts the bounding boxes and their corresponding object classes simultaneously in an end-to-end manner. Unlike traditional object detection methods that use anchor generation or non-maximum suppressors, DETR uses a fully attention-based mechanism to predict object instances directly from the input image (Carison, et al., 2020).

### 2.7.1 Pipeline of DETR

Three main components are in the DETR architecture, which are a simple Feed-forward Network (FFN), Transformer's Encoder-Decoder, and the Convolutional Neural Network (CNN) backbone (Carison, et al., 2020).

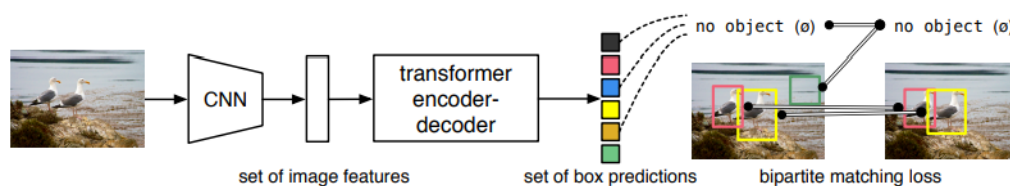


Figure 2.5: Architecture of DETR (Carison, et al., 2020)

At the beginning, the pictures are sent to CNN for feature extraction, like ResNet-50. Typically, the CNN has five max-pooling layers so that it will produce a  $2048 \times W/32 \times H/32$  tensor where H and W are the height and width of the picture respectively (Carison, et al., 2020). However, the Transformer encoders can only process 1D input data instead of 3D input data. So, the feature map tensor needs to pass through a  $1 \times 1$  convolution to become a 2D feature map and then collapse the spatial dimension into a single dimension (Carison, et al., 2020). Thus, the 2D feature map can be transformed into a 1D input data, which is a sequence of tokens that is suitable for Transformer encoders.

Then, since the Transformer encoder cannot recognize the sequence of the tokens, position encoding is added to the flattened feature map tokens before entering the encoder. The encoder in DETR is not so different to the original Transformer Encoder. However, there is a slight difference in the decoder part. In the DETR Decoder, it does not need to do autoregressive like the original Transformer's Decoder. Instead, it just decodes the N objects, which are normally 100, by using parallel decoding at each decoder layer (Carison, et al., 2020).

In order to produce the output embeddings from the DETR Decoder, a fixed number of trainable inputs, which is normally 100 in DETR, are used for the decoder. The trainable inputs are called object queries. These object queries act as examiners and ask every certain region of the picture whether there is an object in the certain region. Same as the encoder, the positional encoding needs to be added to the object queries to differentiate N input embeddings since the decoder is also permutation-invariant (Carison, et al., 2020).

Lastly, DETR sends these output embeddings to the last stations, which are classified FFN and bounding box FFN to classify the object and determine the position of bounding boxes. FNN has a 3-layer perceptron which has a linear projection, a hidden dimension  $d$ , and a ReLU activation function layer. All the normalized width, height and center coordinates of the bounding box are predicted by the FFN, while the class labels are predicted by its linear layer using the softmax function. Given that the fixed-size set of N bounding boxes often significantly exceeds the actual count of objects in the image, the remaining objects with no classes are labeled as 'No Object' class or 'Background' class (Carison, et al., 2020).

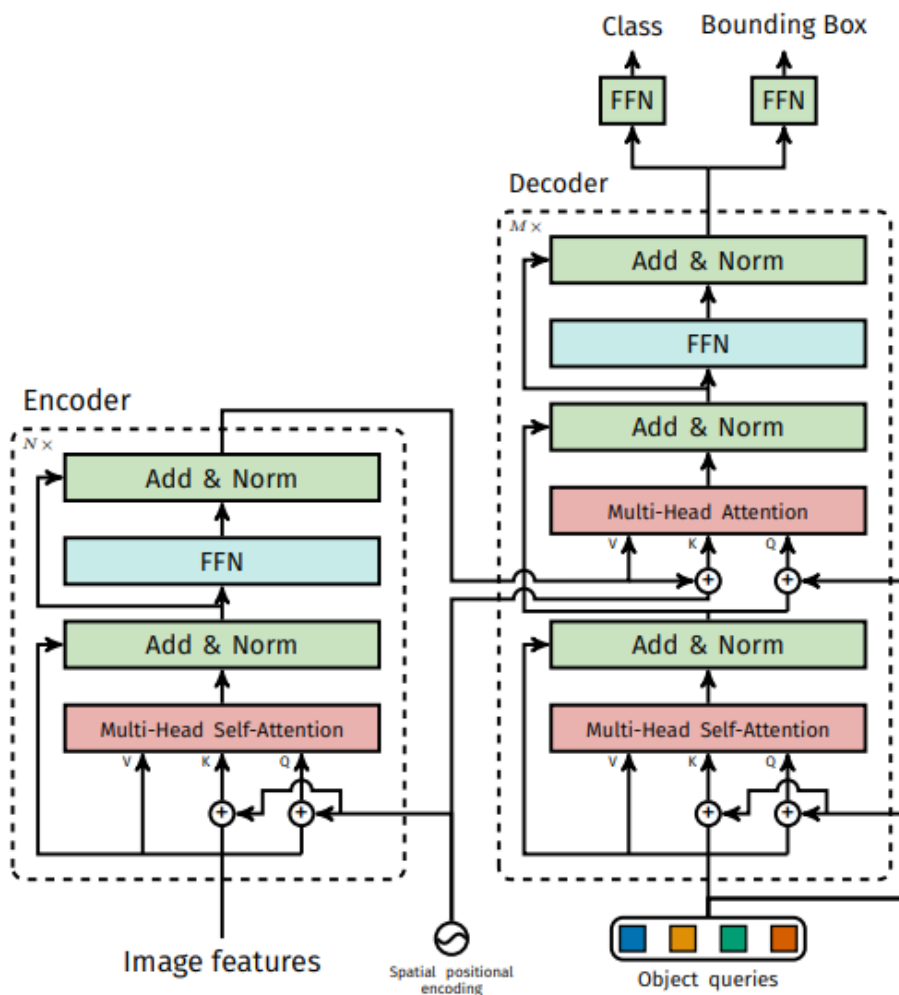


Figure 2.6: Architecture of DETR's Transformer (Carison, et al., 2020)

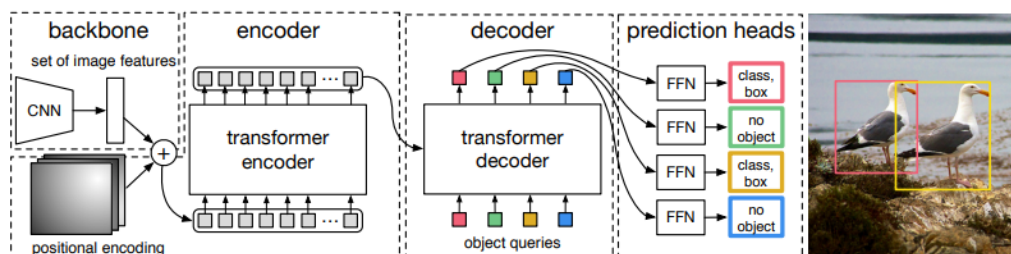


Figure 2.7: Pipeline of DETR (Carison, et al. 2020)

### 2.7.2 Loss Function

DETR is a direct set prediction approach, which means that it must find a one-to-one correspondence between the actual set of objects and the predicted set of objects from the ground truth. Thus, DETR incorporates a matching loss

designed to identify the optimal alignment between the predicted objects and the ground truth objects (Carison, et al., 2020).

The optimal bipartite matching function serves as the loss function, representing a matching between the ground truth sets of objects and the predicted sets of objects, each permuted among  $N$  elements, in a manner that minimizes the associated cost. This function is defined as follows:

$$\hat{\sigma} = \arg \min \sum_i^N \mathcal{L}_{match}(\mathcal{Y}_i, \hat{\mathcal{Y}}_{\sigma(i)}), \sigma \in \mathfrak{S}N \quad (2.3)$$

where

$\mathcal{L}_{match}(\mathcal{Y}_i, \hat{\mathcal{Y}}_{\sigma(i)})$  = pairwise matching cost between ground truth  $\mathcal{Y}_i$  and prediction with index  $\hat{\mathcal{Y}}_{\sigma(i)}$

$$\mathcal{Y}_i = (c_i, b_i)$$

$c_i$  = target class label

$b_i$  = ground truth box position and size (center coordinates  $x, y$ , height and width)

$$\hat{\mathcal{Y}}_{\sigma(i)} = (\hat{c}_i, \hat{b}_i)$$

$\hat{c}_i$  = predicted class

$\hat{b}_i$  = predicted bounding box vector

$\sigma(i)$  = index within permutation of  $N$

The probability of target class is defined as  $\hat{p}_{\sigma(i)}(c_i)$  for the prediction with the index  $\sigma(i)$ , while the predicted bounding box loss is defined as  $\mathcal{L}_{box}(b_i, \hat{b}_{\sigma(i)})$ . Thus, a Hungarian Loss function is formed:

$$\mathcal{L}_{Hungarian}(\mathcal{Y}, \hat{\mathcal{Y}}) = \sum_{i=1}^N [-\log \hat{p}_{\sigma(i)}(c_i) + 1_{\{c_i \neq 0\}} \mathcal{L}_{box}(b_i, \hat{b}_{\sigma(i)})] \quad (2.4)$$

In fact, the log-probability has been downweight by the DETR author by a factor of 10 when the  $c_i = \emptyset$ , which means ‘No Object’ class. This can avoid the imbalance of classes (Carison, et al., 2020).

## 2.8 Real-Time Detection Transformers (RT-DETR)

RT-DETR is a new real-time target detection model from Baidu, and was introduced in a paper 'DETRs Beat YOLOs on Real-time Object Detection' in 2023 (Lv, et al., 2023). It is designed for the implementation of end-to-end real-time object detection with architecture of Transformer. In order to do so, it uses an IoU-aware query selection mechanism and an efficient hybrid encoder, resulting in very high performance in terms of accuracy and inference speed. First, RT-DETR does not require post-processing steps like NMS, which enables it to achieve a stable and efficient inference speed and achieve real-time object detection tasks. Besides that, authors of RT-DETR have optimized the interaction of AIFI and CCFM modules, improving the efficiency of the attention mechanism to achieve faster training convergence. Furthermore, RT-DETR also supports flexible adjustment of inference speed by changing the decoder layer, which allows users to do it without retraining. Not only that, they also provide a few versions of RT-DETR with different parameters and FPS, allowing users to choose the most suitable version of RT-DETR based on the computational power of their training platform.

In the paper, the authors also present empirical evaluations comparing RT-DETR against real-time end-to-end object detectors like YOLO, PPYOLOE, and Efficient-DETR, highlighting RT-DETR's superiority. Based on the paper, RT-DETR-50 achieves an average precision (AP) of 53.1% and operates at 108 frames per second (FPS), while RT-DETR-101 attains 54.3% AP and 74 FPS on COCO val2017. These results surpass YOLO detectors of comparable scale, excelling in both speed and accuracy. Although RT-DETR shows results that may beat YOLO, ongoing research and development in the field of object detection continues, so YOLOv8 was trained in the study as a comparison for RT-DETR.

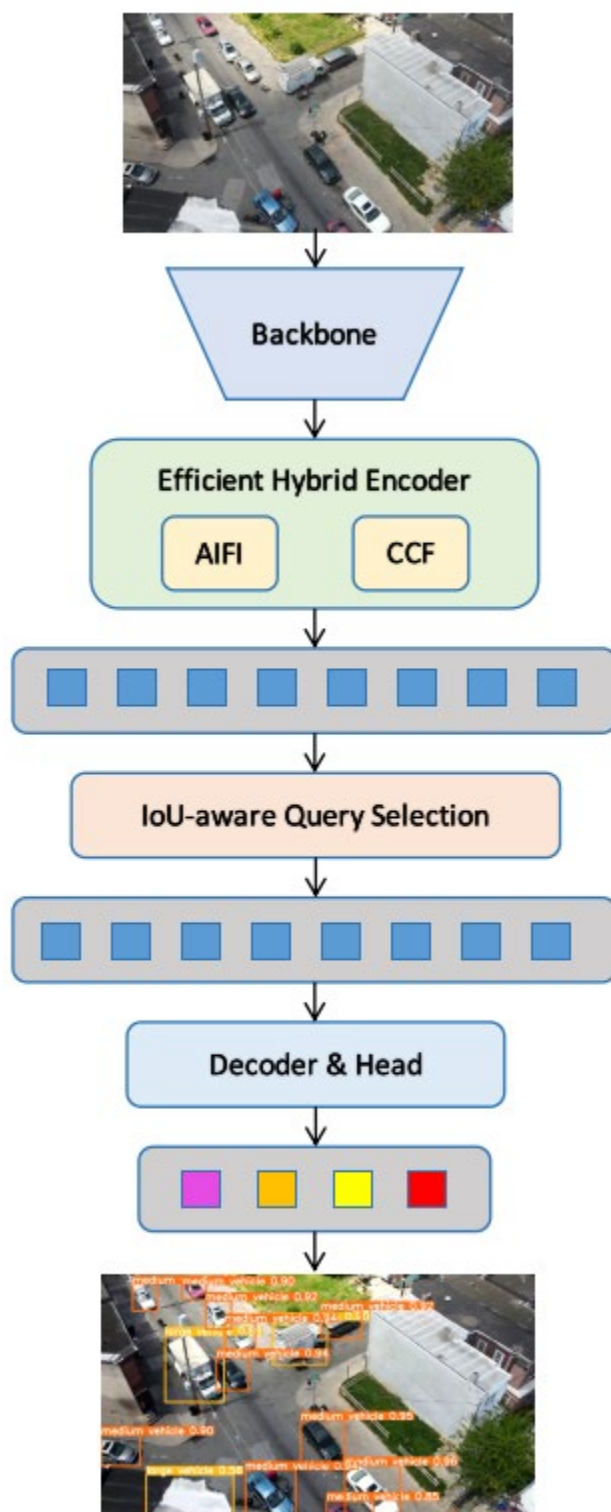


Figure 2.8: Basic Architecture of RT-DETR (Lv, et al. 2023)

## 2.9 Summary

Overall, the literature review conducted in this study has enhanced the understanding of current object detection technologies. The architecture and pipeline of Transformer have been understood clearly and helps the research



smoothly. YOLO is one of the famous traditional CNN approach object detection models. Then, a new object detection technology, DETR (Detection Transformer) has been released which may beat the traditional CNN approach object detection models like YOLO. Thus, a comparative analysis is being conducted between the newest object detection model of the Transformer-based architecture, RT-DETR, and the traditional CNN approach object detection model, YOLOv8, which is the latest YOLO model.

## CHAPTER 3

### METHODOLOGY AND WORK PLAN

#### 3.1 Introduction

This chapter discusses the work plan and the methodology used in this study from the implementation of the models with the required software to hardware. The flow of the implementation starts with a understanding of implementation method, software and hardware used, datasets exploration, models training for object detection and system implementation methodology. Once all the training of models finish, the performance of models are evaluated with some evaluation metrics, which are recall value, mAP (Mean Average Precision), and FPS (Frame Per Second).

#### 3.2 Work Plan

Before conducting the implementation, the flow of the study is planned well and accordingly to ensure the smoothness of the project. At the beginning, the topic and objectives of this study are planned to understand successfully first in order to do further research. Then, based on the topic and objectives, literature review and term explorations will be conducted as planned in the following weeks. During the literature review, some teaching guides on implementation models on the Internet, like YouTube videos, will be watched to learn the way of training the RT-DETR and YOLOv8 models for the custom dataset before conducting the actual training. After understanding the implementation method of the custom dataset training model, the Urban Zone Aerial Object Detection Dataset will be acquired and prepared for model training. Training will commence for both RT-DETR models (with ResNet-18 and ResNet-50 backbones) and YOLOv8 variants (YOLOv8n, YOLOv8s, YOLOv8m) using the same standard dataset. Before start the actual training, all the parameters of models will be fine-tuned first to ensure the models can be trained in the best setting. Once all the training are finished, the models will be evaluated using some evaluation metrics and their results will be recorded. Based on the results, the performance of the models will be compared and discussed, to further analyse their pros and cons. After all the models' performance finish evaluating,

a poster summarizing the study's objectives, methodology, results, and conclusions will be prepared, along with a comprehensive report detailing the research process and findings. Both the poster and report will undergo final review and refinement before submission.

### 3.3 Methodology

The methodology of the study closely follows the outlined work plan. All the planned tasks are completed well before the date of schedule. The details of the training configurations are introduced in a subchapter titled “Training Configuration”. Figure 3.1 shows the overview methodology of the study:

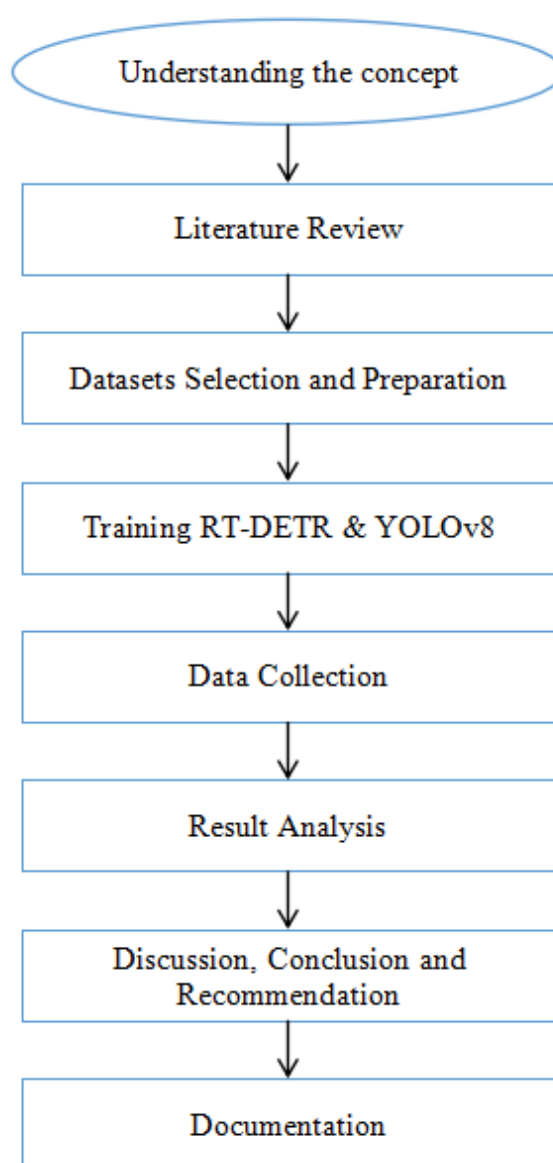


Figure 3.1: Methodology of the Study

### 3.3.1 Similar Studies

In this study, although there are fewer studies directly comparing RT-DETR and YOLOv8, especially in the field of aerial object detection in urban areas, some comparison of similar studies still provides us with some good insights. For example, the study by Bak et al. compared the performance of RT-DETR with other real-time object detection models, revealing its efficacy in coastal debris monitoring (Bak, et al., 2023). Besides that, Aguilera et al. conducted a study focused on evaluating deep learning models, including Mask-RCNN, RT-DETR, and YOLOv7, with the aim of detecting and classifying blueberries (Aguilera, et al., 2024). Some traditional metrics like mean average precision (mAP) and the impact of partial occlusion on models' accuracy have been highlighted in their paper.

### 3.3.2 Hardware

In order to train the object detection models at higher epochs and batch size, a good specification of training platform is required. Hence, the training hardware device used in this study is a NUC (Next Unit of Computing) equipped with an external GPU RTX3080. The choice of NUC is because of its lightweight and easily portable. During the actual testing phase, all the inferences were done by CPU solely without the GPU. This is because the object detection models in this study will usually be installed in mobile devices like drones which lack computational power. This setup helps the inference evaluation be closer to the real-world scenario of real-time inference on drones. Although the inferences are done by CPU only, the training of the models still requires a GPU, which can boost a lot in the training speed, reducing heavy training time. The eGPU RTX3080 of the training platform has a VRAM of 10 GB of memory. It is good enough to train the lightweight object detection models. The specification of the training platform is shown in Table 3.1.

### 3.3.3 Software

In the software, the operating system of the training platform is Linux Ubuntu 20.04. All the training and testing are done on the this Ubuntu 20.04 operating system. It provides a good deep learning environment for training the object

detection models by offering a lot of tools and libraries associated with artificial intelligence. Additionally, Ubuntu also supports some popular frameworks such as TensorFlow, Keras, OpenCV etc.

The main programming languages that are used in this study are Python 3, PyTorch, and PyTorch Lightning. PyTorch is an open-source deep learning framework which was developed by Facebook's AI Research lab (FAIR). It is widely used in the field of artificial intelligence, especially deep learning. This is because it provides a flexible computational graph, which makes it particularly suitable for research and experimentation in artificial intelligence and machine learning (NVIDIA, n.d.).

However, bugs may be introduced when using PyTorch in a complex system or using multi-GPU training. Then, PyTorch Lightning came out. It can solve the problem by structuring the PyTorch code. PyTorch Lightning was created by William Falcon and other professional researchers and PhD students. It is also a lightweight PyTorch that simplifies the deep learning training models. Thus, it can be used to standardize and organize PyTorch code by providing a high-level abstraction and automating various engineering tasks so that the process can be simplified.

Besides that, the GPU Acceleration Library used in the study are CUDA 12.2 and cuDNN 8.8.1. CUDA is used to allow the GPU to do parallel computing while cuDNN is used to optimize the deep neural network in deep learning. Thus, they can improve the efficiency of training and inference. The specification of the training platform is shown in Table 3.1.

Table 3.1: Table of Specification of Training Platform

<b>Name</b>	<b>Specification</b>
<b>Operating System</b>	Ubuntu 20.04
<b>GPU Acceleration Library</b>	CUDA 12.2, cuDNN 8.8.1
<b>CPU</b>	Intel Core i7-10710U CPU @ 4.70GHz
<b>Memory</b>	64 GB
<b>GPU</b>	NVIDIA GeForce RTX 3080 @ 10GB Memory

### 3.3.4 Datasets

The training dataset was Urban Zone Aerial Object Detection datasets and downloaded from the Kaggle. It was combined from three datasets by Sganderla, which are Unmanned Aerial Vehicles Benchmark Object Detection and Tracking (Du, et al. 2018), Vision Meets Drones (Zhu, et al. 2021), and Stanford Drone Dataset (Robicquet, et al. 2016). This dataset has more than 180,000 images, which are split into three groups: "train" (training), "val" (validation), and "test" (testing) files. The "train" file has a total of 131,119 aerial images which are used to train the models. During the training, the model will learn from the "train" (training) dataset, adjusting its parameters to minimize the loss function. Then, the model is evaluated using the validation dataset for every epoch of training, which consists of 28,085 not seen aerial images during training. This validation step is very important as it helps to monitor the model's generalization and prevent overfitting. The performance of the models can be clearly observed every epoch whether the performance of model is getting worse than previous epoch. Lastly, there are 27,934 aerial images in the "test" file. They were used to further verify the models' performance after the completion of training to double confirm the performance of the models. All the training dataset images are captured from urban zones, focusing on four object classes: person, small vehicle, medium vehicle, and large vehicle (Sganderla, 2021).

### 3.3.5 Training Configuration

First, RT-DETR with backbone ResNet-18 was selected as primary model. It is the lightest weight compared to other RT-DETR models. Because of its light weight, it can do the object detection task well without excessive computing overhead. Although the accuracy may decrease compared to heavier models, it can use less computing resources to get the results faster. Hence, it is more well-suited for deployment on resource-constrained devices commonly used in aerial surveillance applications, especially doing real-time object detection tasks. Furthermore, RT-DETR model with a ResNet-50 backbone was also trained to provide a comparative analysis against RT-DETR-r18 and YOLOv8 models, serving as a heavy model counterpart. On the other hand, YOLOv8n, YOLOv8s, and YOLOv8m were chosen as the YOLO compared models as they represent the three lightest variants in the YOLOv8 series, allowing for a comprehensive comparison across the YOLOv8 architecture spectrum. Table 3.2 shows the detailed information of each model parameters and FLOPs.

Table 3.2: Training Models Information

<b>Model</b>	<b>FLOPs(B)</b>	<b>Parameters(M)</b>	<b>Size (pixels)</b>
YOLOv8n	8.7	3.2	640
YOLOv8s	28.6	11.2	640
YOLOv8m	78.9	25.9	640
RT-DETR-r18	60	20	640
RT-DETR-r50	136	42	640

In the training process, all the models were done training on a NUC equipped with an eGPU RTX3080. All the images are resized to 640x640 pixels to ensure consistency during the training. In general, a larger batch size of training can result in improved training accuracy and quicker convergence, but it also introduces a higher risk of overfitting. Based on the research, YOLOv8 is good training in batch size of 16 while RT-DETR is batch size of 4. However, due to

insufficiency of GPU memory, only the YOLOv8m was unable to train in the batch size of 16. Thus, YOLOv8m was trained in batch size of 8. Each model was trained for 100 epochs equally to learn features relevant to object detection in urban environments. After the training, all the trained models were evaluated using the test image dataset, assessing their performance based on the recall values, mAP (Mean Average Precision), and inference speed or FPS (Frame Per Second). Based on the result of each model, a discussion of the result was done to analyze each model's performance and the capability for real-world deployment. Based on the research, the best optimizer for YOLO is SGD while the RT-DETR is AdamW. Because of different optimizer, the initial learning rate is also different. Table 3.3 shows the training parameters in each model.

Table 3.3: Configuration of Training Parameters

<b>Model</b>	<b>Initial Learning Rate</b>	<b>Learning Rate Scheduler</b>	<b>Batch Size</b>	<b>Optimizer</b>
YOLOv8n	0.01	0.01	16	SGD
YOLOv8s	0.01	0.01	16	SGD
YOLOv8m	0.01	0.01	8	SGD
RT-DETR-r18	0.0001	1	4	AdamW
RT-DETR-r50	0.0001	1	4	AdamW



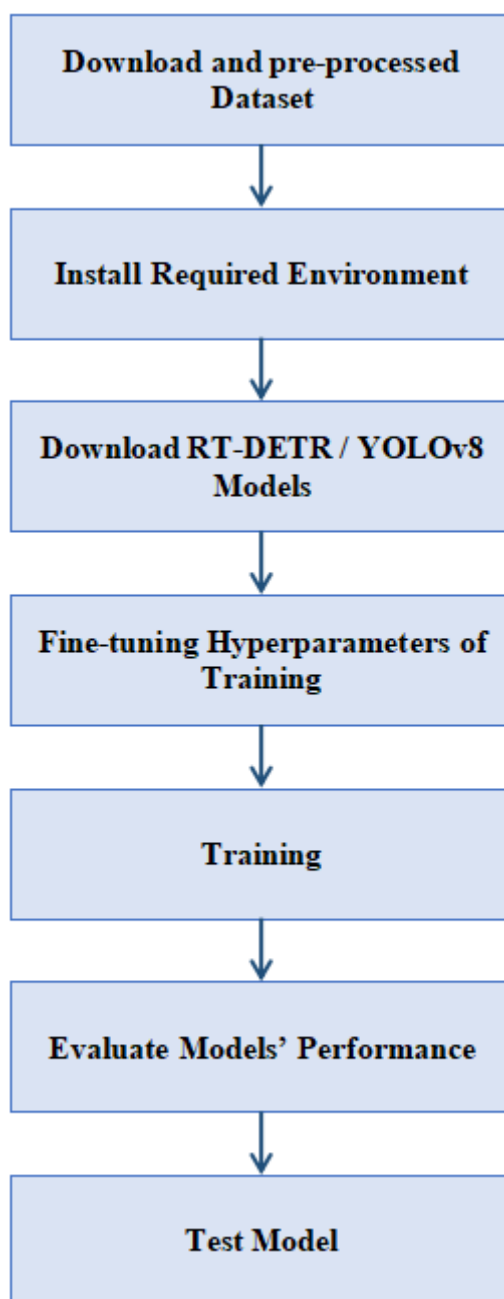


Figure 3.2: Flowchart of Models' Training

### 3.3.6 Testbed

To simulate real-world deployment scenarios, in the evaluating phase, the inference tasks were conducted without the eGPU RTX 3080, relying solely on the CPU for processing. The performance of all the models were evaluated using the test image datasets. The evaluation metrics used in the evaluating phase are Mean Average Precision at an Intersection over Union (IoU) threshold of 0.5 (mAP(50)), mAP(50-95). Recall (R) and Frames per Second (FPS).

Furthermore, a real-world urban zone video was used to further evaluate the models' performance. This video was sourced from YouTube (CharlieBo313, et al., 2021). The video was trimmed and left the first 30 seconds for testing purposes. In the video, the feature footage was captured from a drone showcasing a street view in Philadelphia. There are four classes of objects: persons, small vehicles, medium vehicles, and large vehicles. The evaluation tests were conducted in five models: YOLOv8n, YOLOv8s, YOLOv8m, RT-DETR-r18, and RT-DETR-R50 to assess their performance with this specific video.

### **3.4 Unified Pipeline of Real-Time Object Detection**

The unified pipeline for aerial object detection in urban zones begins with the intake of video sources, which includes aerial footage of urban areas. These videos are then fed into RT-DETR or YOLOv8 object detection models, which can identify and locate objects of interest, such as persons, small vehicles, and large vehicles. Following the detection, the models extract the bounding boxes corresponding to their identified objects. Then, a confidence threshold of 0.5 is applied to the objects detected. If models detect the object with a confidence score above the threshold, its bounding box and corresponding class label (e.g., person, small vehicle, or large vehicle) will be displayed. On the other hand, objects with confidence scores below 0.5 will be ignored, and their bounding boxes and class labels not shown. The whole detecting process continues for each detected object, until the video ends. The output video with annotated bounding boxes and class labels is then saved and presented as output. Not only that, this pipeline also supports video streams as input so that the models can do real-time object detection and tracking in urban zones.

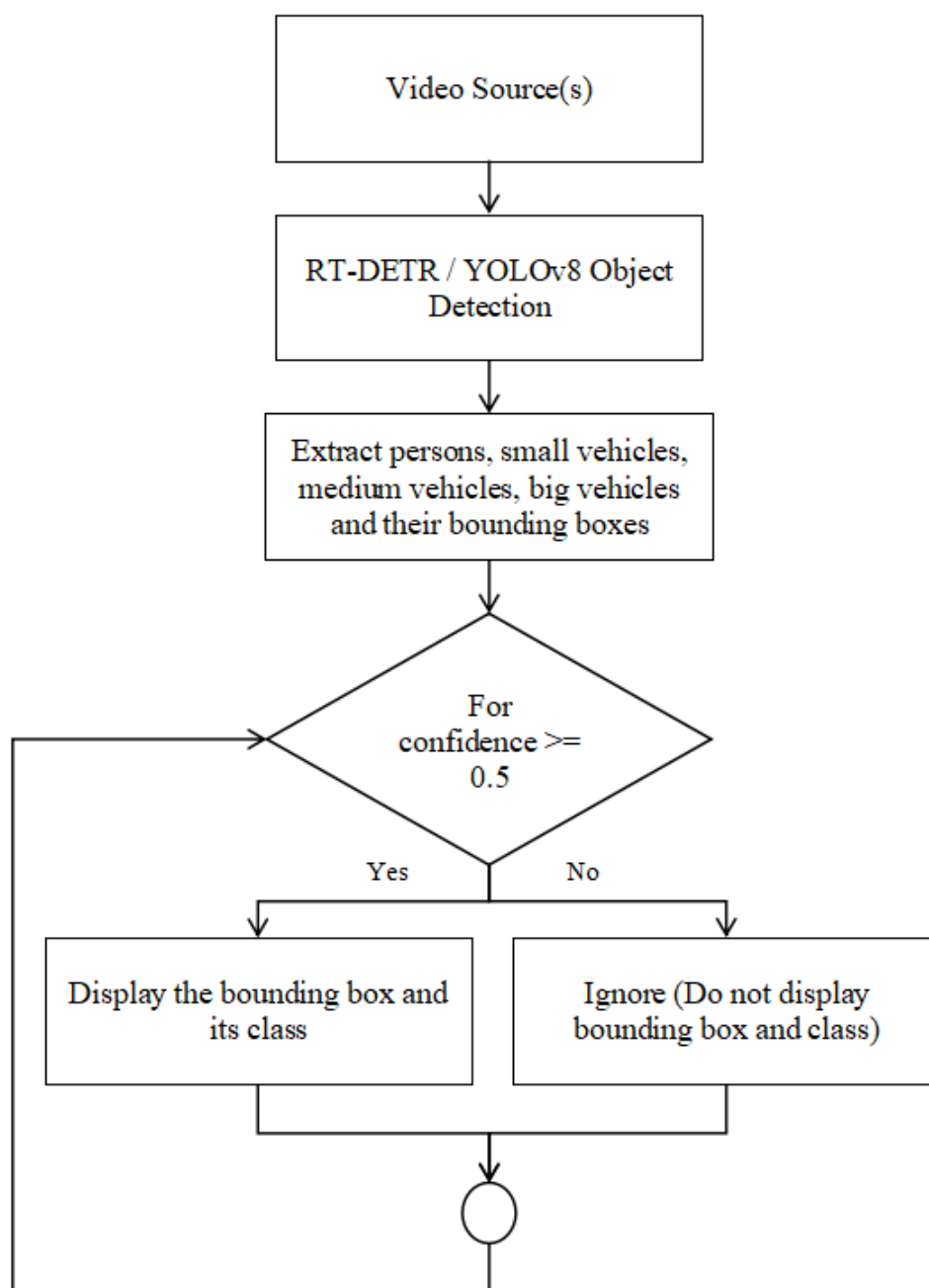


Figure 3.3: Unified Pipeline of Real Time Aerial Object Detection

### 3.5 Gantt Chart

Timeline: 26<sup>th</sup> June 2023 to 19<sup>th</sup> September 2023

Gantt Chart																
No.	Project Activities	Planned Completion Date	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10	W11	W12	W13	W14
1	Understand topic and objectives	01/07/2023	■	■												
2	Literature review	02/09/2023			■	■	■	■	■	■	■	■	■	■		
3	Preliminary testing	13/09/2023							■	■	■	■	■	■	■	
4	FYP 1 progress report writing	15/09/2023						■	■	■	■	■	■	■	■	■
5	Oral presentation preparation	19/09/2023													■	■
6	FYP 1 progress report submission	18/09/2023														■

Figure 3.4: Gantt Chart Phase 1

Timeline: 29<sup>th</sup> January 2024 to 24<sup>th</sup> May 2023

Gantt Chart																
No.	Project Activities	Planned Completion Date	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10	W11	W12	W13	W14
1	Learning RT-DETR	11/02/2024	■	■												
2	Traning RT-DETR and YOLOv8	15/03/2024		■	■	■	■	■	■	■	■					
3	Preparation Conference Paper	22/03/2024				■	■	■	■	■	■	■				
4	Evaluation Testing of Models									■	■	■	■	■	■	
5	Data Analysis	29/03/2024									■	■	■	■	■	
6	FYP 2 Poster Preparation	26/04/2024										■	■	■	■	
7	FYP 2 Final Report Writing	19/04/2024											■	■	■	■
8	FYP 2 Presentation Preparation	03/05/2024													■	■
9	FYP 2 Final Report Submission	03/05/2024														■

Figure 3.5: Gantt Chart Phase 2

### 3.6 Summary

As a short summary, the work plan and the methodology have been successfully implemented. There are 5 models being implemented in this study, which are YOLOv8n, YOLOv8s, YOLOv8m, RT-DETR-r18, and RT-DETR-r50. The dataset used for training is Urban Zone Aerial Object Detection Dataset, which is downloaded from Kaggle. This dataset is used to detect four classes, which are person, small vehicles, medium vehicles, and large vehicles. It has more than 100k aerial images, which are sufficient to train the models. All the training models are trained in a NUC with RTX 3080 eGPU. The operating system of

the training platform is Ubuntu 20.04. All the models are trained in 100 epochs to ensure consistency. After all the models finish training, they are evaluated for their performance using evaluation metrics such as recall value, mAP (Mean Average Precision) and FPS (Frame Per Second). Lastly, these models were further evaluated with a urban zone street view video to further verify their real-world performance.

## CHAPTER 4

### RESULTS AND DISCUSSION

#### 4.1 Introduction

After all the RT-DETR and YOLOv8 models were done training, they were tested by doing inference tasks on the new test image dataset to evaluate the performance of the models. Then, the models were also further assessed by doing inference tasks on a street view video to further evaluate their real-world performance. There are some evaluation metrics used during this phase. They are recall value, mAP(50), mAP(50-95) and FPS (Frame Per Second). The performance results of the models are discussed in this chapter.

#### 4.2 Evaluation Metrics

There are few evaluating metrics used in this study. First, one of the metrics is recall rate (R). Recall (R) is a metric that shows how often the models identify the true objects among all the current true samples. It is get by dividing the number of true positive detections by the total number of ground truth objects in the dataset. Conversely, precision (P) measures the proportion of correctly identified positive objects out of all objects identified as positive by the model. It is calculated by dividing true positives by the sum of false positives and true positives. Both Recall (R) and Precision (P) can be expressed as follows:

$$R = \frac{TP}{(TP + FN)} \quad (4.1)$$

$$P = \frac{TP}{(TP + FP)} \quad (4.2)$$

where,

R : Recall

P : Precision

TP : True Positive

FN : False Negative

FP : False Positive

Average Precision (AP) can be said as the most important metric for evaluating the object detection model's performance. It is determined by computing the area below the precision-recall curve, where precision values are plotted against corresponding recall values at varying confidence thresholds. In object detection, The  $mAP(50)$  shows a comprehensive assessment of the model's accuracy. It considers a detection to be accurate if the IoU between the ground truth and the predicted bounding box equals or more than 0.5.

$$AP = \int_0^1 P(R) dR \quad (4.3)$$

$$mAP(50) = \frac{1}{N} \sum_{i=1}^N AP_i \quad (4.4)$$

where,

$N$  : Number of Object Class

On the other hand,  $mAP(50-95)$  is also the  $mAP$  but the range of IoU thresholds starts from 0.5 to 0.95, with an increment of 0.05. It corresponds to the mean AP for over 10 IoU levels from 0.5 to 0.95 with a step size of 0.05 (0.5, 0.55, ..., 0.95). Since higher IoU thresholds require stricter alignment between predicted and ground truth bounding boxes, achieving high precision becomes more challenging. By getting all the average precision values across this range, it shows a more overall assessment in the performance of models across various confidence levels of overlapping between predicted and ground truth bounding boxes.

$$mAP(50 - 95) = \frac{1}{N} \sum_{i=1}^N AP(50 - 95)_i \quad (4.5)$$

Lastly, FPS (Frame Per Second) in this study represents the frame generated by the models every second. Thus, it is also considered as the model's inference speed. During the inference, when a model needs to use a lot of time to do the inference for one frame, then its FPS will become lower. When a model uses less time to make the inference, it can generate the output frame faster.

Hence, its FPS becomes higher. Therefore, the higher the FPS, the faster the model's inference speed, the smoother the output video generated. During the evaluation, FPS is calculated by using 1000 divided by the total of the time of preprocess, inference, and postprocess.

$$FPS = \frac{1000 \times Frame(s)}{Second(s)} \quad (4.6)$$

### 4.3 Performance of Models in Evaluation Test

Once all the models are finished training, they are being evaluated their performance using the test image dataset. This test image dataset has more than 27k images and is different from the training image dataset. Thus, it is good to use to evaluate the models' performance. The results of the evaluation are collected in Table 4.1. This table compares the performance of each different object detection model, including YOLOv8n, YOLOv8s, YOLOv8m, RT-DETR-r18, and RT-DETR-r50 on the task of aerial object detection in urban zones. Obviously, RT-DETR-r50 gets the best results among all the models with a recall value of 0.86, mAP(50) of 0.904, and mAP(50-95) of 0.598. Furthermore, RT-DETR-r18 exhibited promising results comparable to YOLOv8m, yet with recall value of 0.83, mAP(50) of 0.874, mAP(50-95) of 0.560 and recall value of 0.82, mAP(50) of 0.880, mAP(50-95) of 0.592 respectively. On the other hand, YOLOv8n yielded the lowest performance metrics among the models, getting recall value of 0.70, mAP(50) of 0.756, and mAP(50-95) of 0.440. Meanwhile, YOLOv8s showed an enhanced performance with recall value of 0.78, mAP(50) of 0.840, and mAP(50-95) of 0.529.

Additionally, among the detected object classes, the medium vehicle class consistently achieved the highest mAP across all five models, followed by large vehicle, small vehicle, and person classes. Specifically, the mAP(50) of the medium vehicle class exceeded 0.9 in all models: 0.926 in YOLOv8n, 0.948 in YOLOv8s, 0.959 in YOLOv8m, 0.960 in RT-DETR-r18, and 0.966 in RT-DETR-r50. However, for the person and small vehicle classes, only YOLOv8m, RT-DETR-r18, and RT-DETR-r50 achieved an mAP(50) of over 0.8. The mAP(50) for the person class in the five models were: 0.593 in YOLOv8n, 0.735 in YOLOv8s, 0.803 in YOLOv8m, 0.825 in RT-DETR-r18, and 0.872 in RT-



DETR-r50. For the small vehicle class, the mAP(50) was: 0.643 in YOLOv8n, 0.769 in YOLOv8s, 0.829 in YOLOv8m, 0.809 in RT-DETR-r18, and 0.854 in RT-DETR-r50. Regarding the large vehicle class, only YOLOv8n fell slightly below 0.9 with an mAP(50) of 0.861. The mAP(50) of other models were: 0.909 in YOLOv8s, 0.928 in YOLOv8m, 0.904 in RT-DETR-r18, and 0.922 in RT-DETR-r50. Detailed mAP(50) results for each object class in every model are presented in Table 4.2.

Table 4.1: Results of Each Models

Models	Architecture	Metrics		
		R	mAP(50)	mAP(50-95)
YOLOv8	n	0.70	0.756	0.440
	s	0.78	0.840	0.529
	m	0.82	0.880	0.592
RT-DETR	r18	0.83	0.874	0.560
	r50	<b><u>0.86</u></b>	<b><u>0.904</u></b>	<b><u>0.598</u></b>

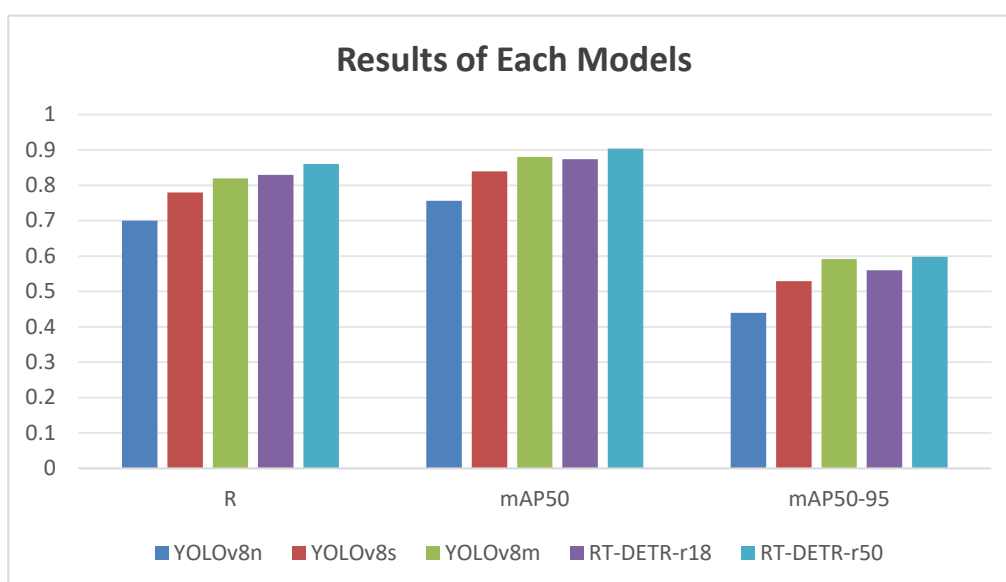


Figure 4.1: Chart of Results of Each Models

Table 4.2: mAP(50) of Object Classes in Each Model

Models	mAP(50)			
	Person	Small Vehicle	Medium Vehicle	Large Vehicle
<b>YOLOv8n</b>	0.593	0.643	<u><b>0.926</b></u>	0.861
<b>YOLOv8s</b>	0.735	0.769	<u><b>0.948</b></u>	0.909
<b>YOLOv8m</b>	0.803	0.829	<u><b>0.959</b></u>	0.928
<b>RT-DETR-r18</b>	0.825	0.809	<u><b>0.960</b></u>	0.904
<b>RT-DETR-r50</b>	0.872	0.854	<u><b>0.966</b></u>	0.922

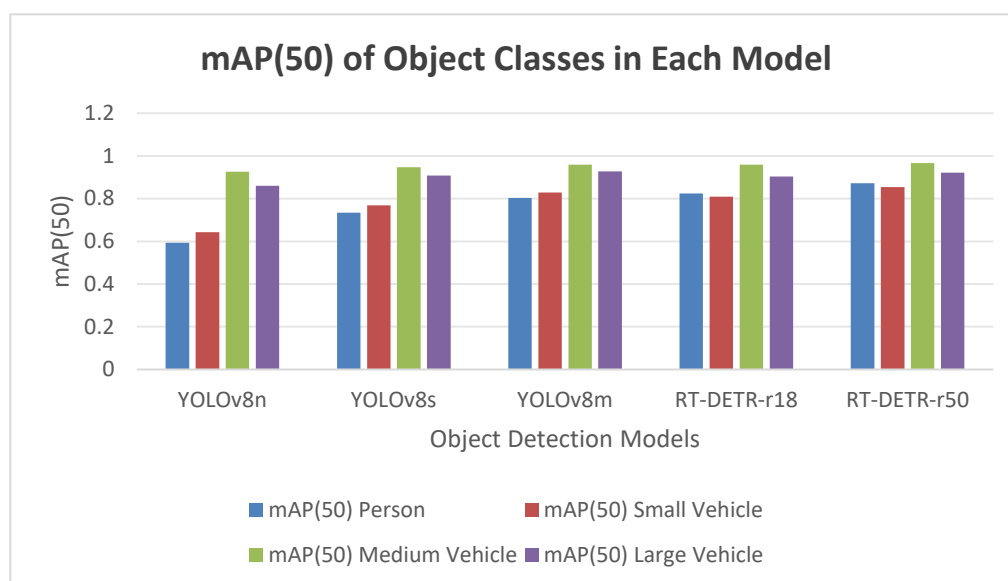


Figure 4.2: Chart of mAP(50) of Object Classes in Each Model

#### 4.4 Performance of Models in Video Processing

Besides the evaluation test using the test image dataset, the models were also implemented to do object detection tasks on a video. This video was sourced from YouTube, which captured a street view of Philadelphia from a drone. For testing purposes, the video was trimmed and left the first 30 seconds. This video contains various objects such as persons, small vehicles, medium vehicles, and large vehicles, which align with the object classes of the trained models. Hence, this video provided a suitable real-world scenario to assess the models' performance.

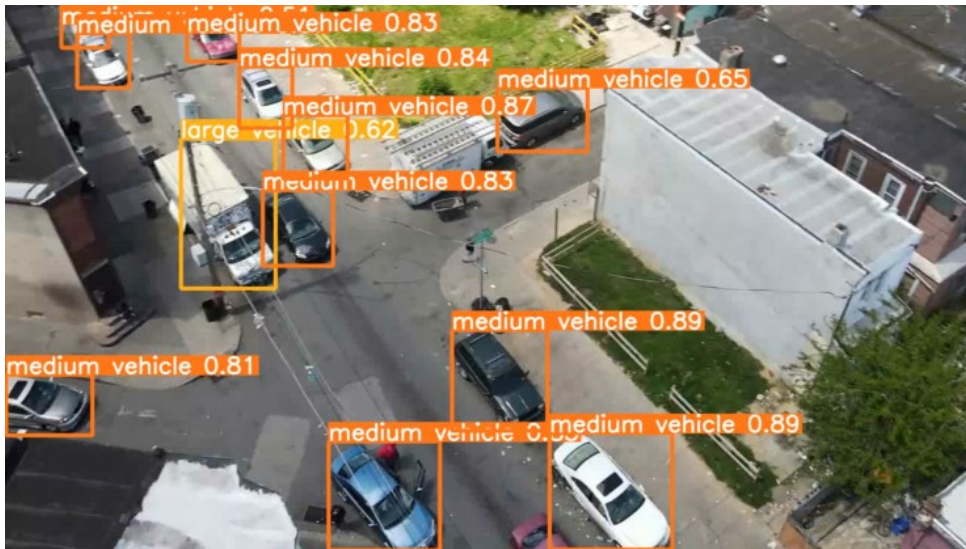


Figure 4.3: YOLOv8n Aerial Object Detection in Video

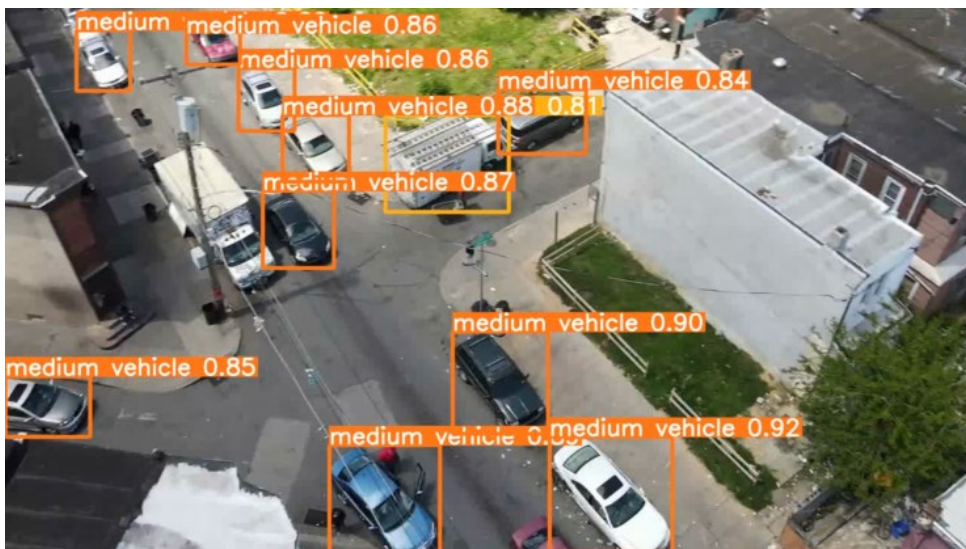


Figure 4.4: YOLOv8s Aerial Object Detection in Video



Figure 4.5: YOLOv8m Aerial Object Detection in Video



Figure 4.6: RT-DETR-r18 Aerial Object Detection in Video



Figure 4.7: RT-DETR-r50 Aerial Object Detection in Video

During the testing of model performance on the video, most of the main objects were successfully detected. The most accurate of detected object classes in the 5 models is medium vehicle class, then followed by large vehicle, small vehicle and person. All the models could detect medium vehicles well, even the lightest weight YOLOv8n. However, in YOLOv8n and YOLOv8s, the small vehicle and person are not detected so perfectly. Sometimes, the models detect the small vehicles as person or detect the person as small vehicles. This problem also occurred in other models but with less frequency. In YOLOv8m, RT-DETR-r18, and RT-DETR-r50, the models were mostly able to detect persons and small vehicles, but errors were still present at times. For the large vehicle class, YOLOv8n and YOLOv8s can detect a few large vehicles only, while YOLOv8m, RT-DETR-r18 and RT-DETR-r50 can detect all the large vehicles that appear in the video. However, RT-DETR-r18 and RT-DETR-r50 sometimes misidentify objects as large vehicles. Overall, the medium vehicles can be detected well in all the five models. If the task is just detecting medium vehicles on a street view, YOLOv8n and YOLOv8s are good enough to do so. For detecting persons, small vehicles, and large vehicles, YOLOv8m, RT-DETR-r18, and RT-DETR-r50 are preferable choices.

Besides that, the FPS of each model was evaluated in this video testing. The FPS was calculated by using 1000 divided by the total of the time of preprocess, inference, and postprocess. After calculating, YOLOv8n had the

highest FPS among all the models, which was 30.4 FPS. This was followed by YOLOv8s, YOLOv8m, RT-DETR-r18, and RT-DETR-r50. Their FPS are 11.4, 5.5, 4.0, and 1.7 respectively. The RT-DETR-r50 had the lowest FPS among all the models. The results of FPS are recorded in Table 4.2.

Table 4.3: FPS in Each Model

Models	Architecture	FPS
<b>YOLOv8</b>	n	<b><u>30.4</u></b>
	s	11.4
	m	5.5
<b>RT-DETR</b>	r18	4.0
	r50	1.7

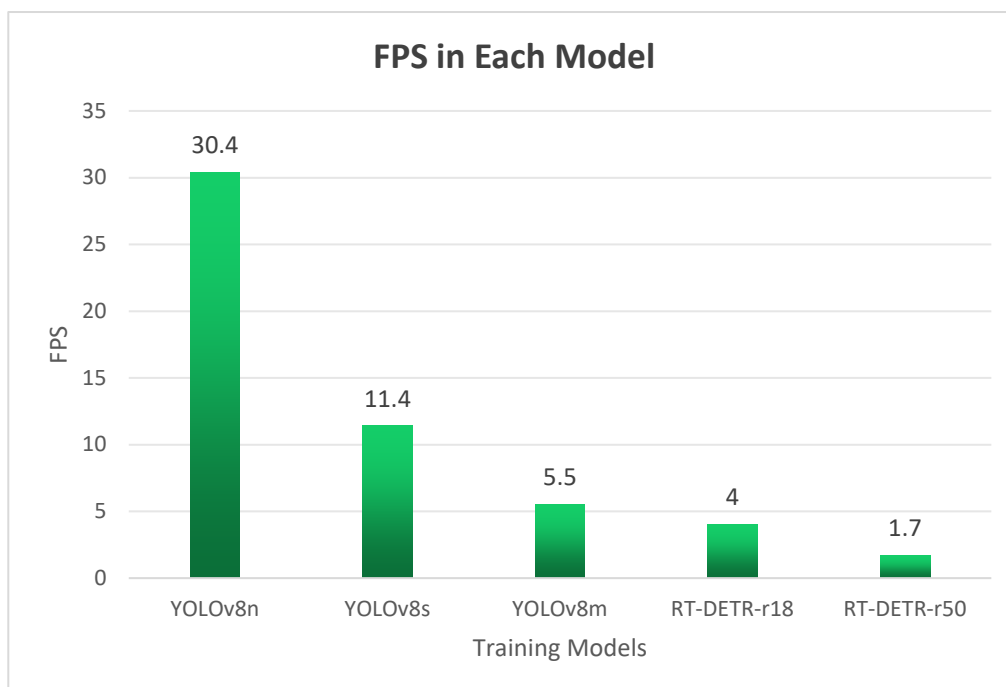


Figure 4.8: Chart of FPS in Each Model

#### 4.5 Impact of FPS in Real-Time Object Detection

On the video processing, all the output videos of models are smooth even if their FPS is low. This is because the models can do the inference tasks for all the

frames of the video, albeit taking longer compared to models with higher FPS. However, when FPS is low during real-time object detection tasks, the models' inference speed may not be fast enough to follow the speed of video capturing. As a result, the bounding boxes may blink, appear and disappear on the targeted objects. This makes it more challenging for users to track moving objects, especially considering these real-time object detection models are typically installed on moving drones. Given that drones are in motion, all objects captured by the drone appear to be moving as well.

#### **4.6 Analysis on the Evaluation Results**

According to the evaluation results, the RT-DETR-r50 has the highest mAP and recall value but the lowest in FPS. This is because the RT-DETR-r50 is the heaviest model compared to others, it has more parameters and FLOPs, resulting in more computational resources being required to progress. Hence, it can detect objects more accurately but needs a longer time during the inference. This phenomenon is also observed in all other models. The heavier the model size, the longer the time taken for inference. Although the heavier models can give a better performance in accuracy, the FPS of them are low due to their slower inference speed. As a result, the YOLOv8n and YOLOv8s act as lightweight models, their FPS could get higher than heavier models like YOLOv8m, RT-DETR-r18, and RT-DETR-r50. Thus, YOLOv8n and YOLOv8s are more preferable in real-time object detection applications since their FPS are high enough to do real-time inference. However, the most recommended model for real-time object detection applications is YOLOv8s as it has high accuracy performance and sufficient inference speed. Because of poor accuracy, the YOLOv8n is not a good choice for object detection tasks. On the other hand, although heavier models like YOLOv8m, RT-DETR-r18 and RT-DETR-r50 are not suitable for real-time object detection applications, they are still preferable, especially in standard object detection tasks which focus on accuracy rather than real-time processing speed.

Besides that, the reason why the medium vehicle class had the highest mAP value is because of the dataset. The training dataset, Urban Zone Aerial Object Detection Dataset used for training, contains a larger number of medium vehicle objects compared to other classes. This larger number of medium

vehicles in the training dataset allows the training models to learn effectively and perform well in inference of medium vehicles. However, the performance of the person and small vehicle classes was not as strong, especially the YOLOv8n and YOLOv8s. This may be because the dataset contains fewer people and small vehicle objects, causing the models to not train enough for inference of these two classes. Another reason may be the person and small vehicle objects are relatively small in the training image dataset. Because of their small size, the training models are hard to extract useful feature maps from the targeted objects, impacting their ability to train effectively. Furthermore, the number of training epochs could not be high enough due to the time constraints. The models have not yet reached the optimum performance. Luckily, this can be easily solved in the future by increasing the training epochs in future studies.

#### **4.7 Summary**

Overall, all the RT-DETR and YOLOv8 models have their own pros and cons. RT-DETR models show comparable results to YOLO models. Although the lightweight models such as YOLOv8n and YOLOv8s show a supremacy in FPS performance, only the YOLOv8s has good performance in accuracy. Hence, among all the five models, only YOLOv8s are well-suited to real-time object detection applications. On the other hand, the heavyweight models like YOLOv8m, RT-DETR-r18 and RT-DETR-r50 are preferable for standard object detection tasks which prioritize accuracy over real-time processing speed. The trade-off between recall value, precision score, and FPS becomes evident, the strategy of selecting models are needed to be determined based on the specific scenario requirements. Last but not least, RT-DETR and YOLOv8 models can be further integrated to get better performance, such as increasing the training epochs, changing the model's architecture, fine-tuning the training hyperparameters, etc.



## CHAPTER 5

### CONCLUSIONS AND RECOMMENDATIONS

#### 5.1 Conclusion

In conclusion, the study successfully implemented two different models, RT-DETR and YOLOv8, for aerial object detection. All the objectives of the study have been achieved successfully. According to the results, YOLOv8s is the most preferable real-time object detection model as it has high accuracy performance (mAP 50-95 of 0.529) and sufficient inference speed (11.4 FPS) in real-time object detection. In contrast, heavier models like YOLOv8m, RT-DETR-r18, and RT-DETR-r50 are not suitable for doing real-time detection tasks as their inference speeds are not fast enough to capture the speed of video capturing. Although like that, the RT-DETR-r50 and RT-DETR-r18 models achieve high mAP scores which are almost the same as YOLOv8m. Therefore, RT-DETR showcases its potential for accuracy, which may beat YOLO, as indicated by the title of the RT-DETR paper, "DETRs Beat YOLOs on Real-time Object Detection". RT-DETR, although its inference speed is slower than YOLO, its Transformer architecture shows a huge potential compared to traditional CNN approaches in object detection, especially in accuracy performance. Therefore, object detection models of Transformer's architecture still have high development possibilities. They are just around the corner.

#### 5.2 Recommendations for future work

Although the RT-DETR and YOLOv8 models have good performance in accuracy, their performance can still be further improved. With the recent release of YOLOv9, YOLOv8 is not the newest model of YOLO anymore. Once the YOLOv9 becomes mature, it should have better performance than YOLOv8. Thus, it is a good choice for training on the YOLOv9 model.

Besides that, because of time constraints, the training epochs of models were set as 100 epochs only. Higher epochs of training are recommended as the models may not have reached their optimal performance. Next, fine-tuning hyperparameters of models is also recommended in future research, like tuning

the learning rate or optimizer. Furthermore, in future exploration, some AI toolkit like OpenVINO is recommended as it can compress and simplify the deep learning models so that these models can be installed in low computational power devices.

Lastly, variant types of DETR models are released currently, like RT-DETR, Efficient DETR, and Deformable DETR. Considering that DETR is an evolving technology, ongoing research and development efforts may introduce new features and optimizations. Thus, it is recommended to do research on a new DETR model once it is released, as it has a huge potential to be evolved.

## REFERENCES

- Aitken, K., Ramesesh, V., Cao, Y. and Maheswaranathan, N., 2021. Understanding How Encoder-Decoder Architectures Attend. arXiv preprint arXiv:2110.15253.
- Agarwal, K., Ashik Sanyo, M. S., Bakshi, S., Vinay, M., Jayapriya, J., & Deepa, S. (2023). Performance Analysis of YOLOv7 and YOLOv8 Models for Drone Detection. 2023 International Conference on Network, Multimedia and Information Technology (NMITCON), 1-10. doi:10.1109/NMITCON58196.2023.10276343.
- Aguilera, C.A., Figueroa-Flores, C., Aguilera, C., & Navarrete, C. 2024. Comprehensive Analysis of Model Errors in Blueberry Detection and Maturity Classification: Identifying Limitations and Proposing Future Improvements in Agricultural Monitoring. *Agriculture*, 14(1), 18. doi: 10.3390/agriculture14010018.
- Alzubaidi, L., Zhang, J., Humaidi, A.J., Al-Dujaili, A., Duan, Y., Al-Shamma, O., Santamaria, J., A.Fadhel, M., Al-Amidie, M., and Farhan, L., 2021. Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. *Journal of Big Data*, 8(5). <https://doi.org/10.1186/s40537-021-00444-8>
- Arkin, E., Yadikar, N., Muhtar, Y. and Ubul, K., 2021. A Survey of Object Detection Based on CNN and Transformer. 2021 IEEE 2nd International Conference on Pattern Recognition and Machine Learning (PRML), pp. 99-108, doi: 10.1109/PRML52754.2021.9520732. IEEE.
- Bahdanau, D., Cho, K. and Bengio, Y., 2016. Neural Machine Translation by Jointly Learning to Align and Translate. arXiv preprint arXiv:1409.0473v7.
- Bak, S. et al. 2023. Applicability Evaluation of Deep Learning-Based Object Detection for Coastal Debris Monitoring: A Comparative Study of YOLOv8 and RT-DETR. *Korean Journal of Remote Sensing*. 대한원격탐사학회, 39(6\_1), pp. 1195 - 1210. doi: 10.7780/KJRS.2023.39.6.1.2
- Brüngel, R. and Friedrich, C., 2021. DETR and YOLOv5: Exploring Performance and Self-Training for Diabetic Foot Ulcer Detection. 2021 IEEE 34th International Symposium on Computer-Based Medical Systems (CBMS), pp. 148-153, doi: 10.1109/CBMS52027.2021.00063. IEEE.
- Carison, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A. and Zagoruyko, S., 2020. End-to-End Object Detection with Transformers. arXiv preprint arXiv:2005.12872.

CharlieBo313. 2021. Philadelphia Most Dangerous Area / Drone Footage. YouTube. Retrieved from <https://youtu.be/nPTO6uxt1aE?si=iBVshvHwTccuiKVV>.

Cordonnier, J., Loukas, A. and Jaggi, M., 2020. On the Relationship between Self-Attention and Convolutional Layers. arXiv preprint arXiv:1911.03584v2.

Dosovitskiy, A., Beyer, B., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., and Jaggi, M., Heigold, G., Gelly, S., Uszkoreit, J. and Houlsby, N., 2021. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. arXiv preprint arXiv:2010.11929v2.

Du, D., Qi, Y., Yu, H., Yang, Y., Duan, K., Li, G., Zhang, W., Huang, Q., Tian, Q. 2018. The unmanned aerial vehicle benchmark: Object detection and tracking. *Proceedings of the European Conference on Computer Vision (ECCV)*, Volume 1, pp. 370-386.

Fang, Y., Liao, B., Wang, X., Fang, J., Qi, J., Niu, J. and Liu, W., 2021. You Only Look at One Sequence: Rethinking Transformer in Vision through Object Detection. arXiv preprint arXiv:2106.00666v3.

Gašparović, B., Mauša, G., Rukavina, J., and Lerga, J., 2023. Evaluating YOLOV5, YOLOV6, YOLOV7, and YOLOV8 in Underwater Environment: Is There Real Improvement?. *2023 8th International Conference on Smart and Sustainable Technologies (SpliTech)*, pp. 1-4, doi: 10.23919/SpliTech58164.2023.10193505. IEEE.

Galvez, R., Bandala, A., Dadios, E., Vicerra, R. and Maningo, J., 2018. Object Detection Using Convolutional Neural Networks, *TENCON 2018 - 2018 IEEE Region 10 Conference, Jeju*, pp. 2023-2027. [https://doi: 10.1109/TENCON.2018.8650517](https://doi.org/10.1109/TENCON.2018.8650517).

Gehring, J., Auli, M., Grangier, D., Yarats, D. and Dauphin, Y., 2017. Convolutional Sequence to Sequence Learning. arXiv preprint arXiv:1705.03122v3

He, K., Zhang, X., Ren, S., and Sun, J., 2016. Deep Residual Learning for Image Recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770-778, doi: 10.1109/CVPR.2016.90.

Hussain, M., 2023. YOLO-v1 to YOLO-v8, the Rise of YOLO and Its Complementary Nature toward Digital Manufacturing and Industrial Defect Detection. *Machines* , 11(7): 677. <https://doi.org/10.3390/machines11070677>

Jason, J., Anderies, Leonico, K., Islamey, J. and Iswanto, I., 2022. Investigating The Best Pre-Trained Object Detection Model for Flutter Framework. *2022 IEEE International Conference on Internet of Things and Intelligence Systems (IoT&IS)*, pp. 235-239, doi: 10.1109/IoT&IS56727.2022.9976010.

Jocher, G., Chaurasia, A., Stoken, A., Borovec, J., NanoCode012, Kwon, Y., Michael, K., TaoXie, Fang, J., imyhxy, Lorna, Zeng, Y., Wong, C., Abhiram V, Montes, D., Wang, Z., Fati, C., Nadar, J., Laughing, ... Jain, M., 2022. ultralytics/yolov5: v7.0 - YOLOv5 SOTA Realtime Instance Segmentation (v7.0). Zenodo. <https://doi.org/10.5281/zenodo.7347926>

Kumar, A. and Srivastava, S., 2020. Object Detection System Based on Convolution Neural Networks Using Single Shot Multi-Box Detector. *Procedia Computer Science*, 171, pp. 2610-2617. <https://doi.org/10.1016/j.procs.2020.04.283>.

Li, Z., Tian, X., Liu, X., Liu, Y. and Shi, X., 2022. A Two-Stage Industrial Defect Detection Framework Based on Improved-YOLOv5 and Optimized-Inception-ResnetV2 Models. *Applied Sciences* 2022, 12(2): 834. <https://doi.org/10.3390/app12020834>

Liu, Y., Han, T., Ma, S., Zhang, J., Yang, Y., Tian, J., He, H., Li, A., He, M., Liu, Z., Wu, Z., Zhu, D., Li, X., Qiang, N., Shen, D., Liu, T. and Ge, B., 2023. Summary of ChatGPT/GPT-4 Research and Perspective Towards the Future of Large Language Models. arXiv preprint arXiv:2304.01852v3.

Luong, M., Pham, H. and Manning, C., 2015. Effective Approaches to Attention-based Neural Machine Translation. arXiv preprint arXiv:1508.04025v5.

Lv, W., Zhao, Y., Xu, S., Wei, J., Wang, G., Cui, C., Du, Y., Dang, Q. and Liu, L., 2023. DETRs Beat YOLOs on Real-time Object Detection. arXiv preprint arXiv:2304.08069v2.

NVIDIA. n.d. *PyTorch*. Available at: <https://www.nvidia.com/en-us/glossary/data-science/pytorch/> (Accessed: 11 September 2023))

O'Shea, K., and Nash, R., 2015. An Introduction to Convolutional Neural Networks. arXiv preprint arXiv:1511.08458v2

Padala, A. and Malathi, P., 2022. An Optimized Object Detection System using You Only Look Once Algorithm and Compare with Deep Neural Networks with increased, 2022 *International Conference on Sustainable Computing and Data Communication Systems (ICSCDS)*, pp. 528-532, doi: 10.1109/ICSCDS53736.2022.9760988.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury Google, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., Devito, Z., Raison Nabla, M., Tejani, A., Chilamkurthy, S., Ai, Q., Steiner, B. and Facebook, L., 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. arXiv preprint arXiv:1912.01703.

Patel, S., and Patel, A., 2021. In: Joshi, A., Khosravy, M., Gupta, N. (eds) *Machine Learning for Predictive Analysis. Lecture Notes in Networks and Systems*, vol 141. *Springer*. [https://doi.org/10.1007/978-981-15-7106-0\\_52](https://doi.org/10.1007/978-981-15-7106-0_52)

Redmon, J., Divvala, S., Girshick, R. and Farhadi, A., 2016. You Only Look Once: Unified, Real-Time Object Detection. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 779-788, doi: 10.1109/CVPR.2016.91. IEEE.

Ren, S., He, K., Ross, G., and Sun, J., 2016. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. arXiv preprint arXiv:1506.01497v3.

Robicquet, A., Sadeghian, A., Alahi, A., Savarese, S. 2016. Learning social etiquette: Human trajectory understanding in crowded scenes. *European Conference on Computer Vision*, Volume 1, pp. 549-565. doi: 10.1007/978-3-319-46484-8\_33.

Sganderla. 2021. Urban Zone Aerial Object Detection Dataset. Kaggle. Retrieved from <https://www.kaggle.com/datasets/sganderla/urban-zone-aerial-object-detection-dataset>.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A., Kaiser, L and Polosukhin, I., 2023. Attention Is All You Need. arXiv preprint arXiv:1706.03762v7.

Wang, G., Chen, Y., An, P., Hong, H., Hu, J. and Huang, T., 2023. UAV-YOLOv8: A Small-Object-Detection Model Based on Improved YOLOv8 for UAV Aerial Photography Scenarios. *Sensors*, 23(16):7190. <https://doi.org/10.3390/s23167190>

Xiong, R., Yang, Y., He, D., Zheng, K., Zheng, S., Xing, C., Zhang, H., Lan, Y., Wang, L., and Liu, T., 2020. On Layer Normalization in the Transformer Architecture. arXiv preprint arXiv:2002.04745v2.

Xu, R., Lin, H., Lu, K., Cao, L. and Liu, Y., 2021. A forest fire detection system based on ensemble learning. *Forests*, 12(2), p.217.

Yao, Z., Ai, J., Li, B., and Zhang, C. 2021, Efficient DETR: Improving End-to-End Object Detector with Dense Prior. arXiv preprint arXiv:2104.01318.

Zhang, H., Li, F., Liu, S., Zhang, L., Su, H., Zhu, J., Ni, L. and Shum, H., 2022. DINO: DETR with Improved DeNoising Anchor Boxes for End-to-End Object Detection. arXiv preprint arXiv:2203.03605v4 .

Zhao, Z., Zheng, P., Xu, S., and Wu, X., 2019. Object Detection with Deep Learning: A Review. arXiv preprint arXiv:1807.05511v2.

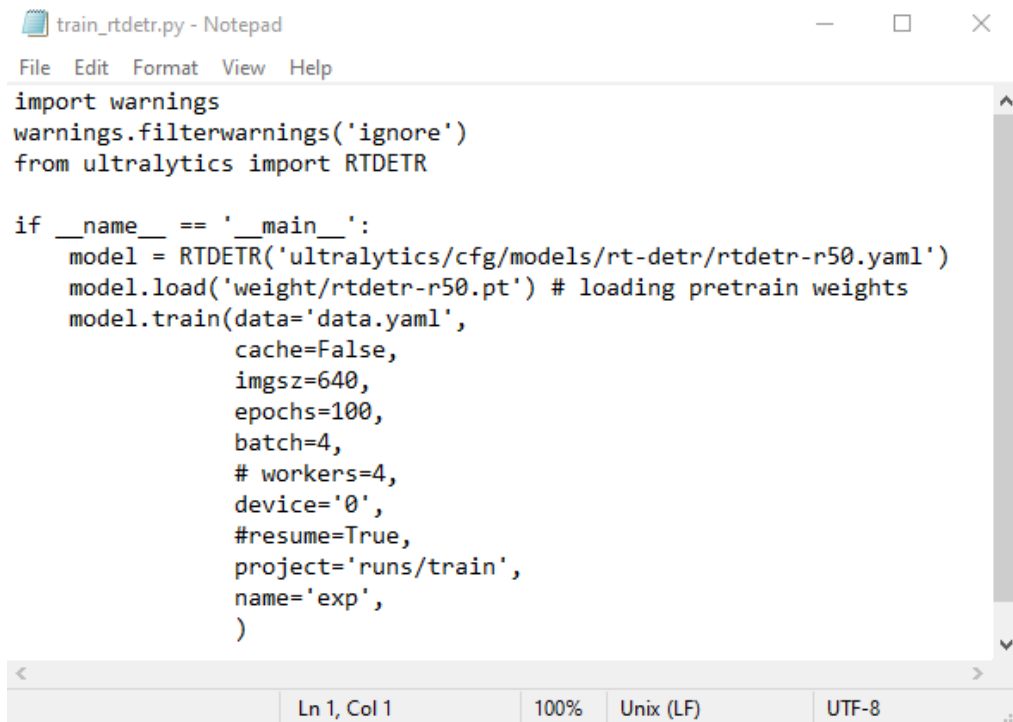
Zheng, D., Dong, W., Hu, H., Chen, X. and Wang, Y., 2023. Less is More: Focus Attention for Efficient DETR. arXiv preprint arXiv:2307.12612.

Zhu, P., Wen, L., Du, D., Bian, X., Hu, Q., Ling, H. 2021. Detection and Tracking Meet Drones Challenge. arXiv preprint arXiv:2001.06303v3.

Zhu, X., Su, W., Lu, L., Li, B., Wang, X., and Dai, J., 2021. Deformable DETR: Deformable Transformers for End-to-End Object Detection. arXiv preprint arXiv:2010.04159v4.

## APPENDICES

### Appendix A: Figures



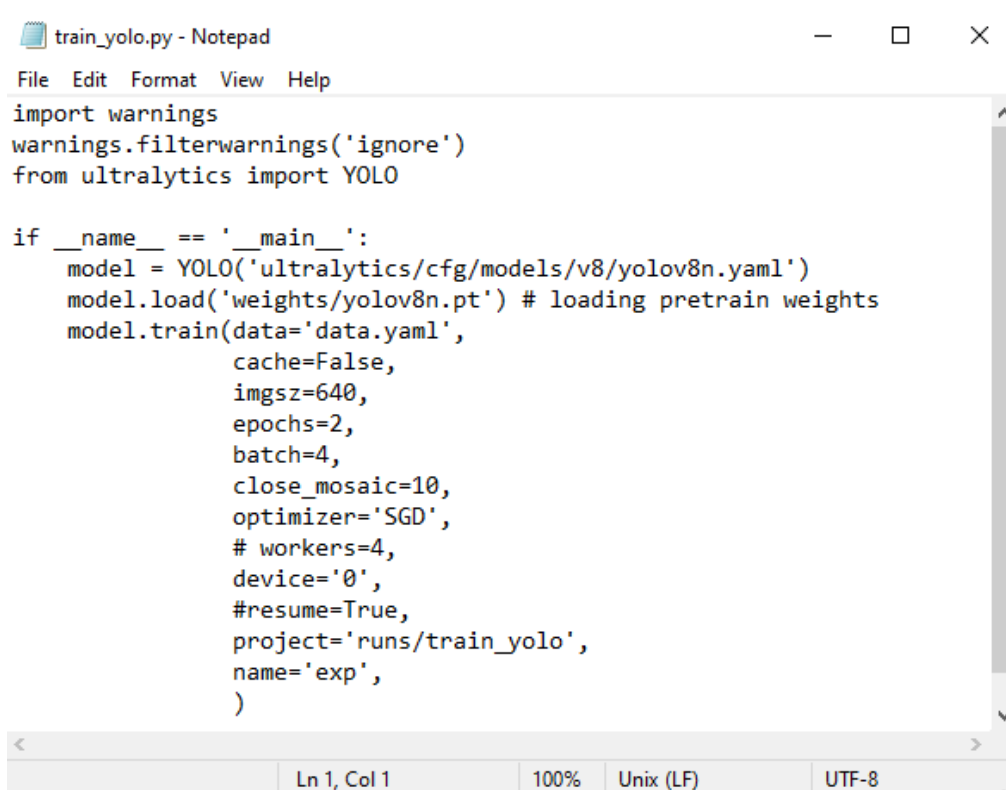
```
train_rtdetr.py - Notepad
File Edit Format View Help
import warnings
warnings.filterwarnings('ignore')
from ultralytics import RTDETR

if __name__ == '__main__':
    model = RTDETR('ultralytics/cfg/models/rt-detr/rtdetr-r50.yaml')
    model.load('weight/rtdetr-r50.pt') # loading pretrain weights
    model.train(data='data.yaml',
                cache=False,
                imgsz=640,
                epochs=100,
                batch=4,
                # workers=4,
                device='0',
                #resume=True,
                project='runs/train',
                name='exp',
                )
```

Ln 1, Col 1    100%    Unix (LF)    UTF-8

Appendix A-1: Training Code of RT-DETR





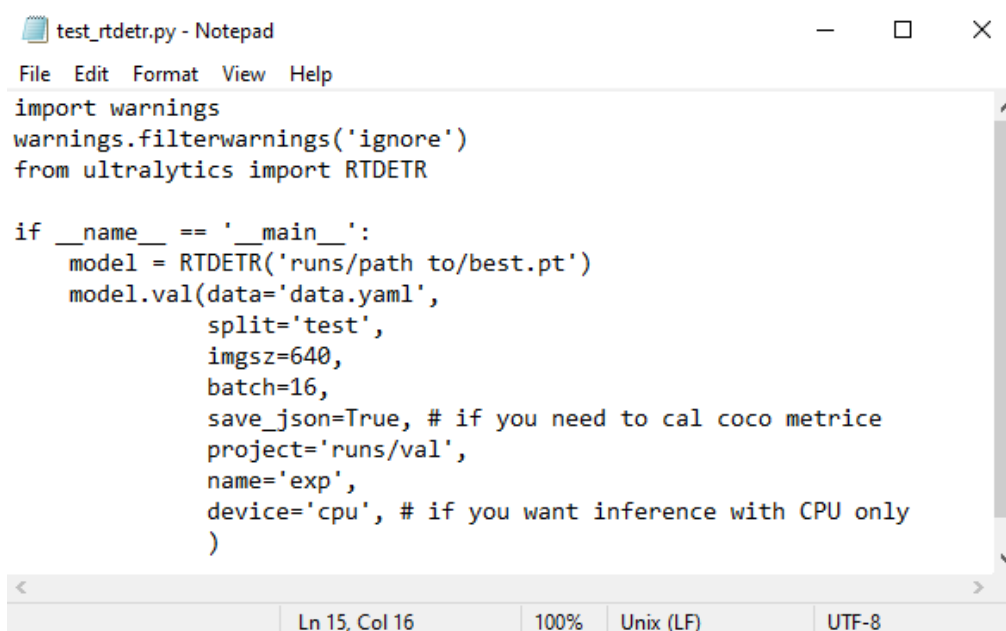
```

train_yolo.py - Notepad
File Edit Format View Help
import warnings
warnings.filterwarnings('ignore')
from ultralytics import YOLO

if __name__ == '__main__':
    model = YOLO('ultralytics/cfg/models/v8/yolov8n.yaml')
    model.load('weights/yolov8n.pt') # loading pretrain weights
    model.train(data='data.yaml',
                cache=False,
                imgsz=640,
                epochs=2,
                batch=4,
                close_mosaic=10,
                optimizer='SGD',
                # workers=4,
                device='0',
                #resume=True,
                project='runs/train_yolo',
                name='exp',
                )
Ln 1, Col 1    100%    Unix (LF)    UTF-8

```

Appendix A-2: Training Code of YOLOv8



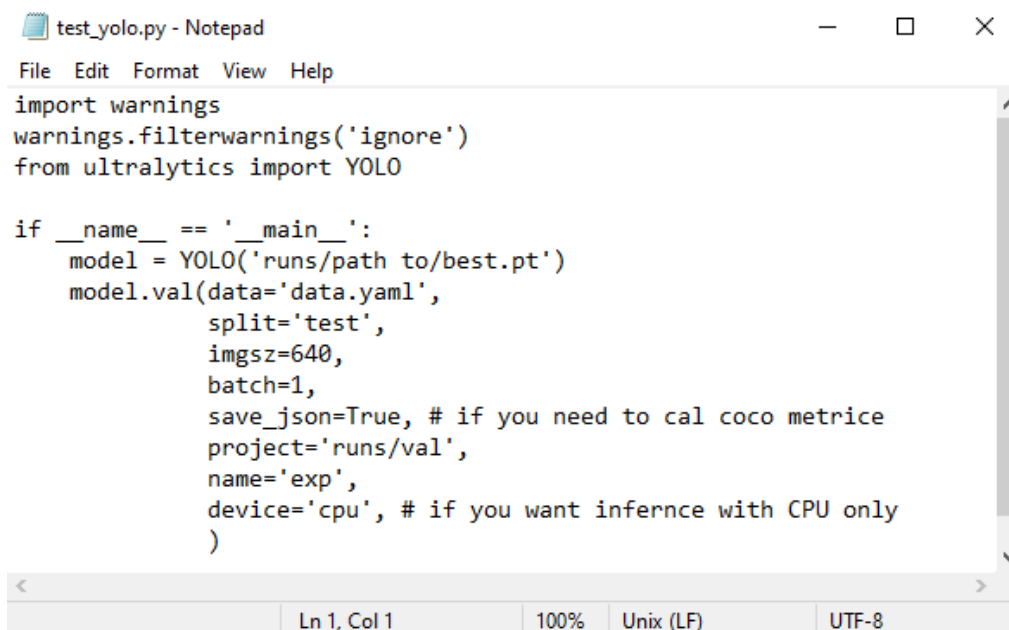
```

test_rtdetr.py - Notepad
File Edit Format View Help
import warnings
warnings.filterwarnings('ignore')
from ultralytics import RTDETR

if __name__ == '__main__':
    model = RTDETR('runs/path to/best.pt')
    model.val(data='data.yaml',
             split='test',
             imgsz=640,
             batch=16,
             save_json=True, # if you need to cal coco metric
             project='runs/val',
             name='exp',
             device='cpu', # if you want inference with CPU only
             )
Ln 15, Col 16    100%    Unix (LF)    UTF-8

```

Appendix A-3: Evaluation Code of RT-DETR on Test Image Dataset



```

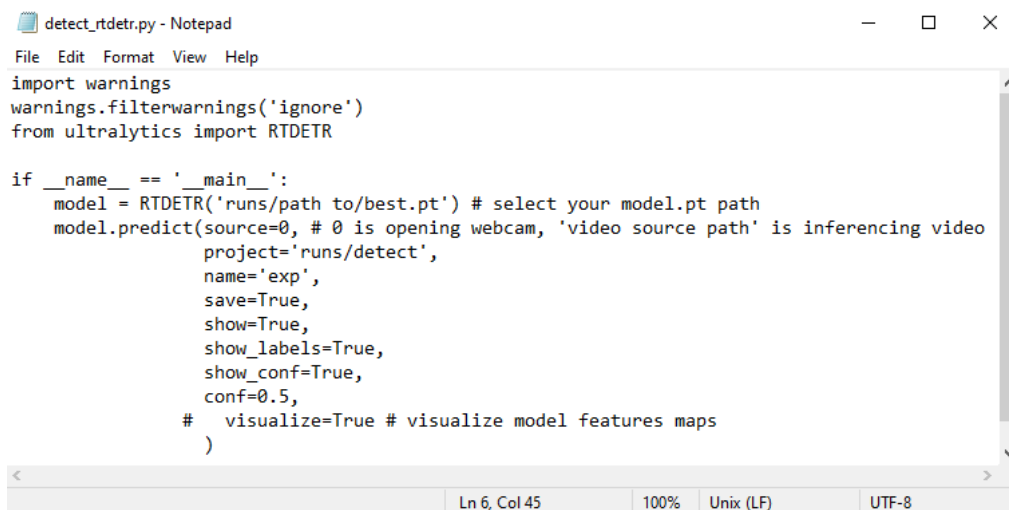
test_yolo.py - Notepad
File Edit Format View Help
import warnings
warnings.filterwarnings('ignore')
from ultralytics import YOLO

if __name__ == '__main__':
    model = YOLO('runs/path to/best.pt')
    model.val(data='data.yaml',
              split='test',
              imgsz=640,
              batch=1,
              save_json=True, # if you need to cal coco metric
              project='runs/val',
              name='exp',
              device='cpu', # if you want inference with CPU only
              )

Ln 1, Col 1    100%    Unix (LF)    UTF-8

```

Appendix A-4: Evaluation Code of YOLOv8 on Test Image Dataset



```

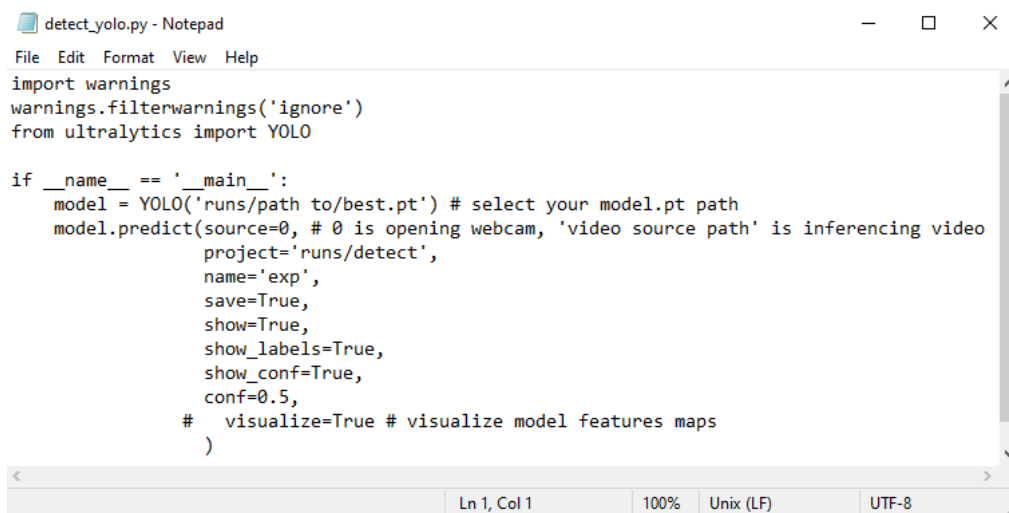
detect_rtdetr.py - Notepad
File Edit Format View Help
import warnings
warnings.filterwarnings('ignore')
from ultralytics import RTDETR

if __name__ == '__main__':
    model = RTDETR('runs/path to/best.pt') # select your model.pt path
    model.predict(source=0, # 0 is opening webcam, 'video source path' is inferencing video
                  project='runs/detect',
                  name='exp',
                  save=True,
                  show=True,
                  show_labels=True,
                  show_conf=True,
                  conf=0.5,
                  # visualize=True # visualize model features maps
                  )

Ln 6, Col 45    100%    Unix (LF)    UTF-8

```

Appendix A-5: Detection Code of RT-DETR on Video



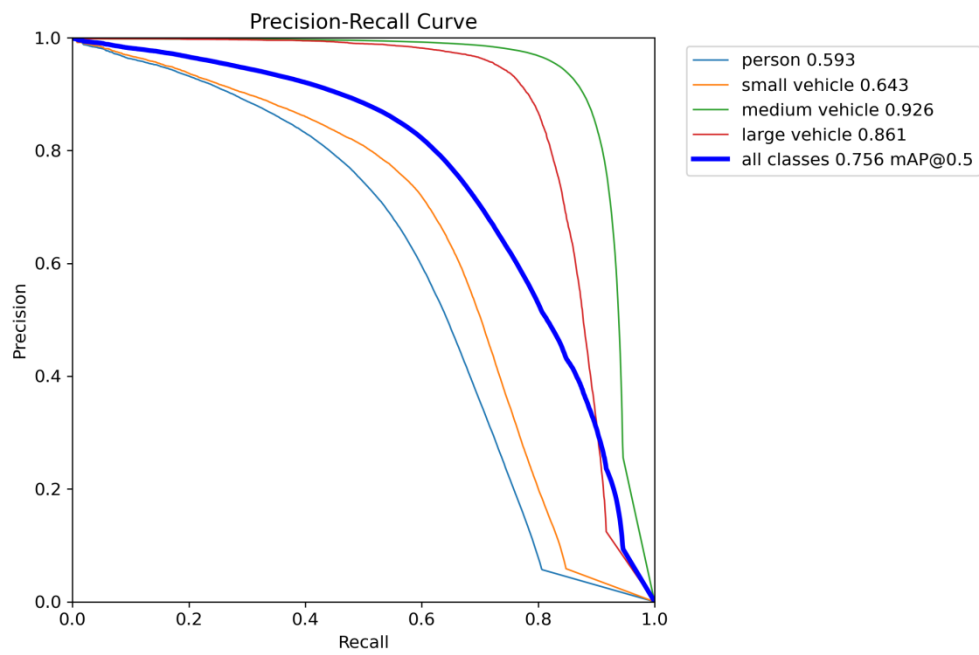
```
detect_yolo.py - Notepad
File Edit Format View Help
import warnings
warnings.filterwarnings('ignore')
from ultralytics import YOLO

if __name__ == '__main__':
    model = YOLO('runs/path to/best.pt') # select your model.pt path
    model.predict(source=0, # 0 is opening webcam, 'video source path' is inferencing video
                  project='runs/detect',
                  name='exp',
                  save=True,
                  show=True,
                  show_labels=True,
                  show_conf=True,
                  conf=0.5,
                  # visualize=True # visualize model features maps
                  )
```

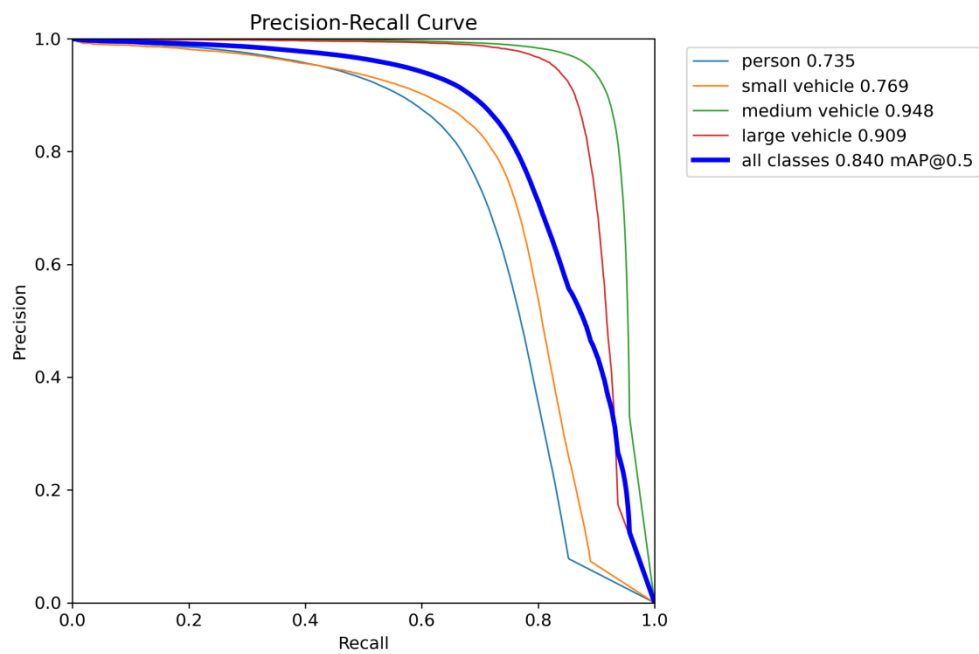
Ln 1, Col 1    100%    Unix (LF)    UTF-8

Appendix A-6: Detection Code of YOLOv8 on Video

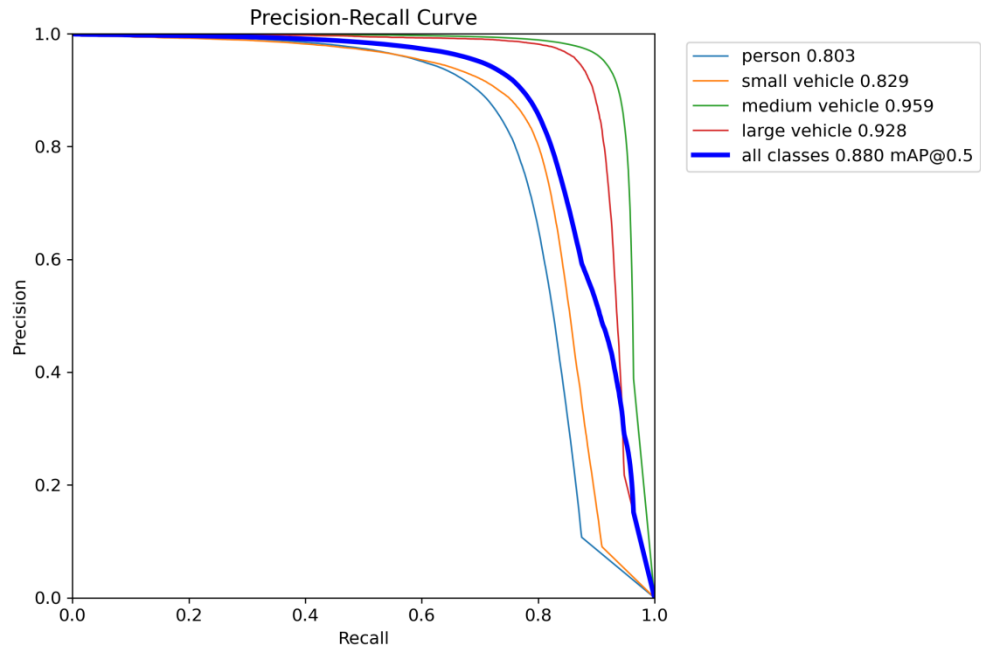
## Appendix B: Graphs



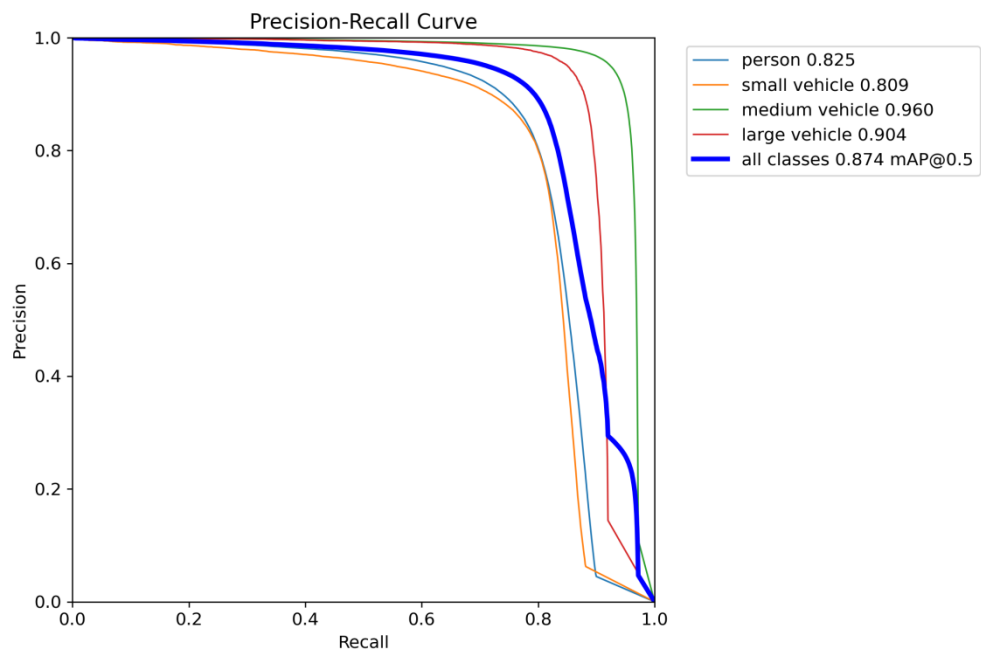
Appendix B-1: Precision-Recall Curve of YOLOv8n



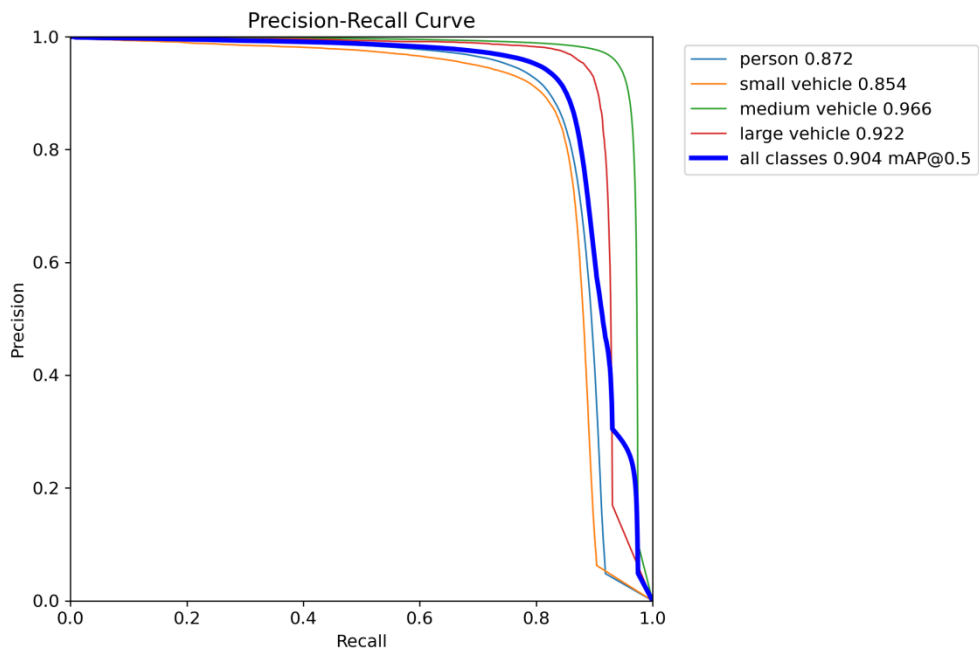
Appendix B-2: Precision-Recall Curve of YOLOv8s



Appendix B-3: Precision-Recall Curve of YOLOv8m



Appendix B-4: Precision-Recall Curve of RT-DETR-r18



Appendix B-5: Precision-Recall Curve of RT-DETR-r50