A

Project Report

On

VEHICLE DETECTION USING YOLO MODEL

Submitted to

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY ANANTAPUR, ANANTAPURAMU

in partial fulfillment of the requirements for the award of the Degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

Submitted by

ANNADANAM PRANAV KARTHIK	**(199E1A05J7)**
T KHUSHI	**(199E1A05M3)**
K HEMANTH	**(199E1A05L9)**
YELLOJI SAI NIKHIL	**(199E1A05P6)**

Under the Guidance of
Mr. J. Shankar Babu, M.Tech.,
Assistant Professor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SRI VENKATESWARA ENGINEERING COLLEGE
(Approved by AICTE, New Delhi, NBA & NAAC Accredited Institution with UGC section 2(f) & 12(b)
& Affiliated to JNTUA, Ananthapuramu)
Karakambadi Road, TIRUPATI – 517507
2019-2023

SRI VENKATESWARA ENGINEERING COLLEGE

(Approved by AICTE, New Delhi, NBA & NAAC Accredited Institution with UGC section 2(f) & 12(b) & Affiliated to JNTUA, Ananthapuramu

Karakambadi Road, TIRUPATI – 517507

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the project report entitled* “VEHICLE DETECTION USING YOLO MODEL” *a bonafide record of the project work done and submitted by

ANNADANAM PRANAV KARTHIK	**(199E1A05J7)**
T KHUSHI	**(199E1A05M3)**
K HEMANTH	**(199E1A05L9)**
YELLOJI SAI NIKHIL	**(199E1A05P6)**

for the partial fulfillment of the requirements for the award of B.Tech Degree in ***COMPUTER SCIENCE AND ENGINEERING****, JNT University Anantapur, Ananthapuramu during the year 2019-2023*

GUIDE	**HEAD OF THE DEPARTMENT**
Mr. J. SHANKAR BABU, M.Tech	**Dr. K. SANTHI**
Assistant Professor	**HOD & Professor**
Department of CSE	**Department of CSE**

External Viva-Voce Exam Held on ______________________________

INTERNAL EXAMINER **EXTERNAL EXAMINER**

DECLARATION

We hereby declare that the project report entitled **“VEHICLE DETECTION USING YOLO MODEL”** done by us under the guidance of **Mr. J. Shankar Babu,** and is submitted in partial fulfillment of the requirements for the award of the Bachelor’s degree in **Computer Science and Engineering.** This project is the result of our own effort and it has not been submitted to any other University or Institution for the award of any degree or diploma other than specified above.

ANNADANAM PRANAV KARTHIK	**(199E1A05J7)**
T KHUSHI	**(199E1A05M3)**
K HEMANTH	**(199E1A05L9)**
YELLOJI SAI NIKHIL	**(199E1A05P6)**

ACKNOWLEDGEMENT

We are thankful to our guide **Mr. J. Shankar Babu** for his valuable guidance and encouragement. His helping attitude and suggestions have helped us in the successful completion of the project.

We would like to express our gratefulness and sincere thanks to **Dr. K. Santhi,** Head of the Department of COMPUTER SCIENCE AND ENGINEERING, for her kind help and encouragement during the course of our study and in the successful completion of the project work.

We have great pleasure in expressing our hearty thanks to our beloved Principal **Dr. C. Chandrasekhar,** for spending his valuable time with us to complete this project. Successful completion of any project cannot be done without proper support and encouragement.

We sincerely thank to the **Management** for providing all the necessary facilities during the course of study.

We would like to thank our parents and friends, who have the greatest contributions in all our achievements, for the great care and blessings in making us successful in allour endeavors.

ANNADANAM PRANAV KARTHIK	**(199E1A05J7)**
T KHUSHI	**(199E1A05M3)**
K HEMANTH	**(199E1A05L9)**
YELLOJI SAI NIKHIL	**(199E1A05P6)**

TABLE OF CONTENTS

ABSTRACT

Every year, the number of vehicles on the road will be increasing. There were around 295 million units of vehicles recorded in India as of December 31, 2019. While as from the mid-2017, there were around 230 million units of motor vehicles. Consequently, accurate and fast detection of vehicles on the road is needed, by using the volume of vehicles as valuable data for detecting traffic congestion which then benefits for traffic management.

Some of the applications are video surveillance, vehicle navigation, intelligent robots control, unmanned vehicles, automated traffic control and biomedical image analysis etc. this project is using TensorFlow which is a platform for machine learning and you only look once (YOLO) which is an object detection algorithm for real-time vehicle detection, By combining these two and other dependencies with python as programming language. The accuracy of detection of vehicles will be increased compared to previous versions.

LIST OF FIGURES

LIST OF ABBREVIATIONS

S.NO	NAME	ABBREVIATION
1	ITS	Intelligent Transport Systems
2	YOLO	You Only Look Once
3	AI	Artificial Intelligence
4	ML	Machine Learning
5	GPU	Graphics Processing Unit
6	CNN	Convolutional Neural Networks
7	IoU	Intersection over Union
8	AP	Average Precision
9	NMS	Non- Maximal Suppression
10	MOT	Motion Object Tracking

CHAPTER - 1
INTRODUCTION

Road accidents are the major cause of death for people between the ages of 5-29 worldwide. More than 1.35 million people lose their lives every year and 50 million are injured in road accidents.

India remains the world's number one for road crash fatalities since 2008. According to the Indian government, about 1.5 lakh of people are killed each year by traffic accidents. These major challenges cannot be addressed only through traditional measures, including the expansion of existing transport infrastructure. Intelligent Transportation System (ITS) have been developed for the purpose of providing efficient services related to the different modes of transport and traffic management without embodying knowledge as such, allowing different users to be better informed and safer more organized and smart use of transport networks.

Number of vehicles on the road are also increasing. There were around 295 million units of vehicles recorded in India as of December 31, 2019. While as from the mid-2017, there were around 230 million units of motor vehicles.

The robust and reliable detection of vehicles is the first course of action for these systems. The identification and monitoring processes of vehicles include high- speed, low-speed, close-distance and self-driving. Over the last 15 years, significant attention has been paid to vision-based vehicle identification for driver assistance due to the dramatic loss in human lives and financial resulting from motor vehicle crashes, the availability of feasible technologies that have been accumulated over the past three decades from computer vision studies and the rapidly growing processor speeds.

In ITS, the vehicle type classification is of the utmost importance because it could be used for anomaly identification, counting, traffic breaches for specific vehicle types, etc.

CHAPTER - 2
LITERATURE SURVEY

[1] Ajinkya Marode, Akash Ambadkar, Aniket Kale, Tilak Mangrudkar(2021) :

- This Object detection system uses YOLO v3.
- It is capable of accurate vehicle detection with near real-time performance.

DRAWBACKS:

- The images in the dataset does not cover all the different environments and lighting conditions.
- This model is also uses the older version of YOLO.

[2] Sumeyye CEPNI, Muhammed Enes ATIK, Zaide DURAN(2020):

- In this paper, The proposed system used You only look once v3 which is unlike the versions that are present at that time.
- The dataset chosen for this system is COCO dataset. It stands for Common Objects in Context. The COCO dataset contains challenging, high-quality visual datasets for computer vision.
- The Accuracy and precision are state of the art at that time.

DRAWBACKS:

- This v3 is released in 2018 which is very old and the current versions are gives much more precise results.
- To attain the accuracy needed Strong training is required, as the primary focus of this system is support Unmanned Aerial Vehicles (UAV).

[3] M. H. Putra, Z. M. Yussof, K. C. Lim, S. I. Salim:

- In this paper, the proposed system uses Yolo algorithm with 7 convolutional layers.
- This system works in such a way that precision increases with the increase in the layers.
- It focuses mostly on detecting cars and persons.

DRAWBACKS:

- As the layers increases complexity and time of processing increases.
- It cannot detect other classes of vehicles.

[4] Haythem Bahri, David Krcma , Jan Koc(2019):

- In this paper, the object detection algorithm is designed for the augmented reality.
- This system aims to help the wearable device user to detect and to recognize between objects in real world. For the object detection approach, a deep learning model has been used for the implementation of this system called YOLO.
- This model is near to real-time and it supports to detect more than 9000 objects.

DRAWBACKS:

- Augmented reality is still in development phase and currently the need of developing autonomous vehicles and traffic management is more important.
- The system is also not producing promising results.

CHAPTER - 3
PROJECT DESCRIPTION

3.1 PROBLEM DEFINITION

Traffic management system has become one of the most researched areas because of the initiatives of Smart Cities and growing popularity of AI and ML. Increased vehicle compactness on roads has resulted in excess of challenges for traffic management in urban cities world. Failure in transporting accident victims, critical patients and medical equipment on time has led to loss of human lives which are the results of road congestions.

Critical traffic problems such as surveillance and traffic congestion require the development of new transportation systems Intelligent Transportation Systems (ITS) aimed at addressing critical issues like traffic congestion and navigation, by integrating different image processing algorithms with machine learning algorithms into transportation and traffic management systems. Due to the recent advancements in image recognition algorithms, You Only Look Once (YOLO) enabled a plethora of applications.

3.2 PROJECT DETAILS

YOLO is a state of the art real time object detection algorithm introduced in 2015 by Joseph Redmon, Santosh Divvala, Ross Girshick and Ali Farhadi in their famous research paper “You Only Look Once: Unified, Real-Time Object Detection”. This algorithm is fast and open source. By combining YOLO with several convolutional neural network layers, detection and classification accuracy can attain state of the art results.

CHAPTER - 4
COMPUTATIONAL ENVIRONMENT

4.1 SOFTWARE SPECIFICATION

• Operating system	: Windows 10.
• Coding Language	: Python.
• Platform	: Google Colab.
• Framework	: TensorFlow.

4.2 HARDWARE SPECIFICATION

• System	: Intel I5.
• Hard Disk	: 40 GB.
• Ram	: 8 GB(min)
• Monitor	: SVGA

4.3 SOFTWARE FEATURES

GOOGLE COLAB

Fig 4.1 Google Colab

Google Colaboratory, or "Colab" as most people call it, is a cloud-based Jupyter notebook environment. It runs in your web browser and lets anyone with internet access experiment with machine learning and coding for artificial intelligence. You can write and execute Python code, share your code and edit it simultaneously with other team members, and document everything by combining it into a single notebook with rich text, charts, images, HTML, and LaTeX.

Artificial intelligence and machine learning: A quick primer

You've heard about artificial intelligence (AI) and have probably heard the term machine learning (ML). While AI and ML are often used interchangeably, ML is a subset or subcategory of Artificial Intelligence. Machine learning is one of the tools or pathways to artificial intelligence, using algorithms to learn insights and recognize patterns from data.

A simple explanation of AI is computer hardware that mimics the capabilities of our own computing hardware, the human brain. By using tools like ML, artificial intelligence gains the ability to learn and make decisions without being explicitly programmed on how to make those decisions or being given all the potential outcomes. Essentially, ML takes the approach of letting a computer learn to program itself through its own experience.

If a company currently deploys AI programs, they use machine learning. ML starts with data — huge amounts of data. The controversial subject of AI-generated art is a good example, as it uses data sampling made up of other people's artwork to train the model. Even with all that data, artificial intelligence still can't paint like a human.

If you are a traditional programmer, you know that programming is like writing cooking recipes for a meal. When programming traditionally, you create detailed instructions telling the computer exactly what to do. The computer follows those instructions. If your code is good, it bakes the same cake you made and wrote the recipe for.

Sometimes writing code for a computer to follow isn't possible or would be so time- consuming that the resources aren't available to do it. There are some tasks that humans can do easily but are difficult to program computers to do, like recognizing people's faces, knowing how to make a piece of art look like Van Gogh painted it, or telling the difference between donuts and bagels. Artificial intelligence is mostly capable of doing these things thanks to machine learning.

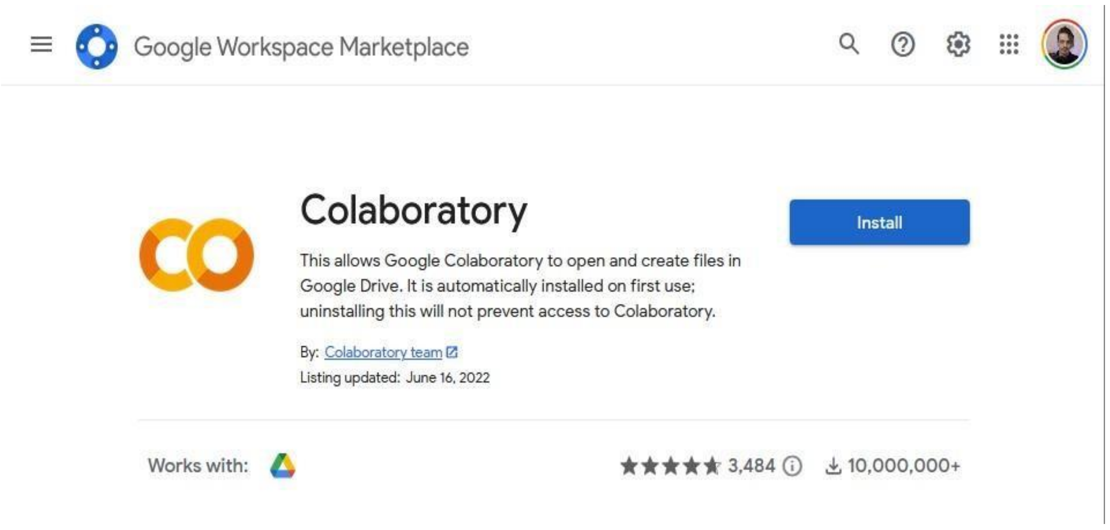
That's artificial intelligence and machine learning in a nutshell. Machine learning lets AI attempt to figure things out by giving it tons of data to learn from. This takes equally huge amounts of computing power to run tests or practice the most basic code. That's where Google Colab comes in.

Why use Google Colab?

Google has been aggressive in the field of AI research. Being a company with enormous resources, it can continually experiment and make breakthroughs in the field of Quantum AI. So, it also has a vested interest in the future of these technologies. Google's AI framework, called TensorFlow, was made open source in 2015. This was followed by making Google's development tool, Colaboratory, free for public use in 2017.

You heard that correctly. You have access to these things right now. Making TensorFlow and Google Colab available to the public has made education about and the development of machine learning applications easier. Even if you can't afford the costly computational infrastructure, you can write and execute code today.

The Google Colab workspace app is installed through the Google Workspace Marketplace and integrates with Google Drive. All of your work is stored in Drive or can be loaded from your GitHub. Everything can be shared using the share settings in Google Drive, Docs, and Sheets. Your code is executed in a virtual machine that is private to your account.



This allows Google Colaboratory to open and create files in Google Drive. It is automatically installed on first use; uninstalling this will not prevent access to Colaboratory.

By: Colaboratory team
Listing updated: June 16, 2022

Fig 4.2 Google Colab extension

Python and Jupyter can have intensive CPU and GPU workload requirements. Colab gives you free access to computing infrastructure to test and execute your code. Like many of Google's products, there is a free tier and paid options. The free version of

Colab is for students, hobbyists, and small experimental projects. As a data scientist or AI researcher, Google's paid plans offer more compute units, faster GPUs, access to higher memory machines, and terminal access with the connected virtual machine.

If you want to learn about artificial intelligence and machine learning or have some simple Python code you want to document or experiment with, Colab requires no setup and is free to use. Google also offers various Colab Pro subscriptions that give paid users access to faster NVIDIA GPUs and compute credits for more advanced tasks.

What can you do in Google Colab?

As a programmer, these are some of the things that are possible in Colab:

- Write, execute, and share code in Python
- Participate in real-time collaborative coding with your team
- Connect with GitHub to import or publish notebooks
- Import external datasets
- Document code that supports mathematical equations
- Access GPU and TPU runtimes for free
- Use preinstalled libraries like TensorFlow, Matplotlib, PyTorch, and other ML libraries
- Integrate with GitHub
- Use version history similar to Google Docs
- Train models using images, audio, and text
- Analyze and visualize data

Google Colab vs. Jupyter Notebook

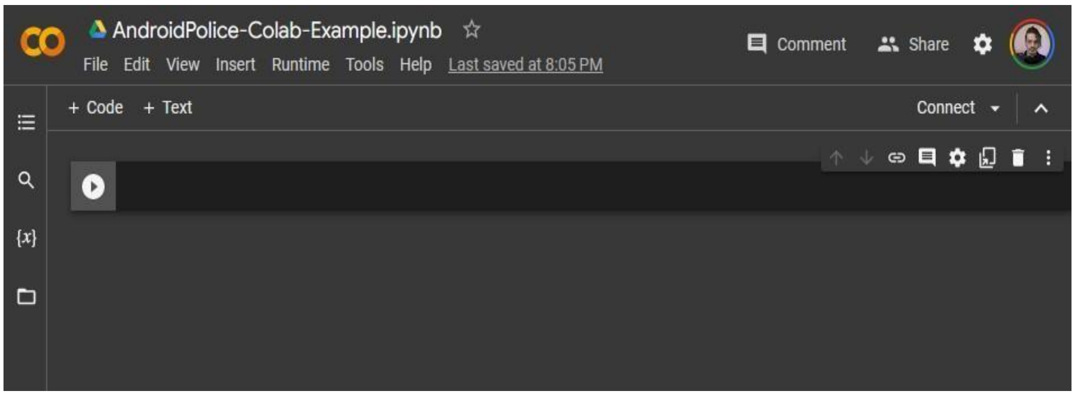

AndroidPolice-Colab-Example.ipynb
File Edit View Insert Runtime Tools Help Last saved at 8:05 PM
Comment
Share
+ Code + Text
Connect

Fig 4.3 Google Colab interface

Google Colab is built on Jupyter Notebook, a fully open source product that's also available for free. Jupyter came first, and IPYNB format notebooks are typically used for data exploration, machine learning experimentation and modeling, documenting code examples, and creating tutorials. Essentially the same things you would do in Google Colab.

So if Google Colab is a way to work with Jupyter Notebooks, what is the difference between using them traditionally or in Google Colab? These are the key differences between the two:

Collaboration tools: The most apparent difference comes down to why Google Colab was named Google Colab. Google's platform provides several tools to make team collaboration easier. Besides document sharing and cloud storage, the most important is real-time collaborative coding with other team members.

Software: Using Jupyter Notebook traditionally requires software installations on your local hardware. You also need to install your own libraries. Colab works 100% in your web browser, so that is the only software you need, and it's software you already have.

Document sharing: Colab notebooks are stored and shared using Google Drive. Like Google Docs and Sheets, your notebooks automatically save periodically, have version history, and can be shared using the same sharing permissions. You can also share your Colab files with anyone without the other person needing to install software to see it.

Computing power: Traditional Jupyter Notebooks are stored locally, and code is executed using your local machine's hardware. Even if you have a blazing-fast home computer, it's limited in comparison to the computing power Google Colab gives you access to.

YOLO (you only look once)

Object detection is a computer vision task that involves identifying and locating objects in images or videos. It is an important part of many applications, such as surveillance, self-driving cars, or robotics. Object detection algorithms can be divided into two main categories: single-shot detectors and two-stage detectors.

One of the earliest successful attempts to address the object detection problem using deep learning was the R-CNN (Regions with CNN features) model, developed by Ross Girshick and his team at Microsoft Research in 2014. This model used a combination of region proposal algorithms and convolutional neural networks (CNNs) to detect and localize objects in images.

Object detection algorithms are broadly classified into two categories based on how many times the same input image is passed through a network.

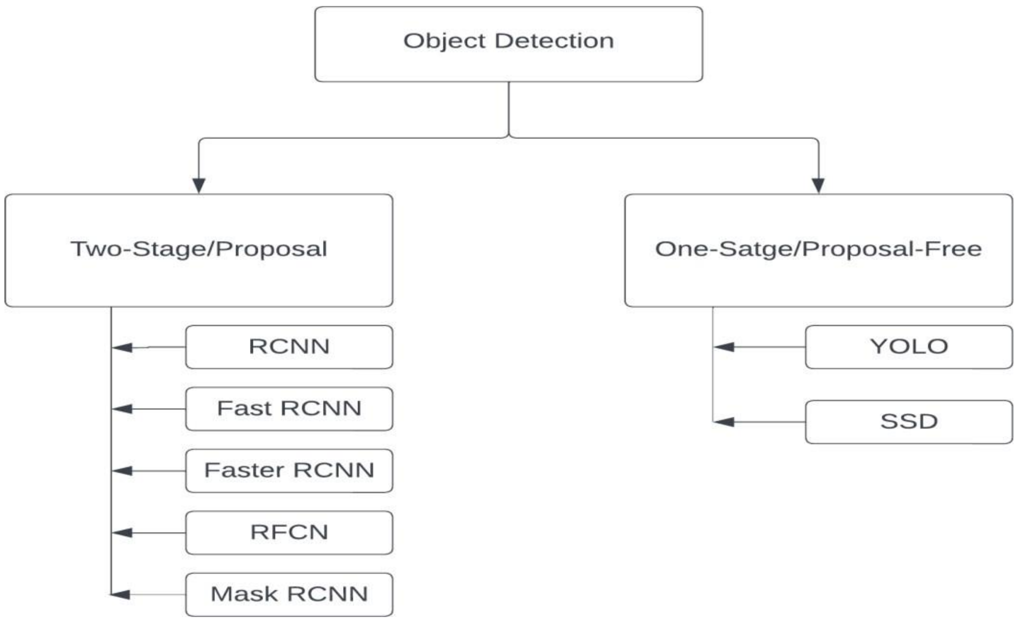


Fig 4.4 Types of Object Detection

Single-shot object detection

Single-shot object detection uses a single pass of the input image to make predictions about the presence and location of objects in the image. It processes an entire image ina single pass, making them computationally efficient.

However, single-shot object detection is generally less accurate than other methods, and it’s less effective in detecting small objects. Such algorithms can be used to detect objects in real time in resource-constrained environments.

YOLO is a single-shot detector that uses a fully convolutional neural network (CNN) to process an image. We will dive deeper into the YOLO model in the next section.

Two-shot object detection

Two-shot object detection uses two passes of the input image to make predictions about the presence and location of objects. The first pass is used to generate a set of proposals or potential object locations, and the second pass is used to refine these proposals and make final predictions. This approach is more accurate than single-shot object detection but is also more computationally expensive.

Overall, the choice between single-shot and two-shot object detection depends on the specific requirements and constraints of the application.

Generally, single-shot object detection is better suited for real-time applications, while two-shot object detection is better for applications where accuracy is more important.

Object detection models performance evaluation metrics

To determine and compare the predictive performance of different object detection models, we need standard quantitative metrics.

The two most common evaluation metrics are Intersection over Union (IoU) and the Average Precision (AP) metrics.

Intersection over Union (IoU)

Intersection over Union is a popular metric to measure localization accuracy and calculate localization errors in object detection models.

To calculate the IoU between the predicted and the ground truth bounding boxes, we first take the intersecting area between the two corresponding bounding boxes for the same object. Following this, we calculate the total area covered by the two bounding boxes— also known as the "Union" and the area of overlap between them called the "Intersection."

The intersection divided by the Union gives us the ratio of the overlap to the total area, providing a good estimate of how close the prediction bounding box is to the original bounding box.

$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}} =$$

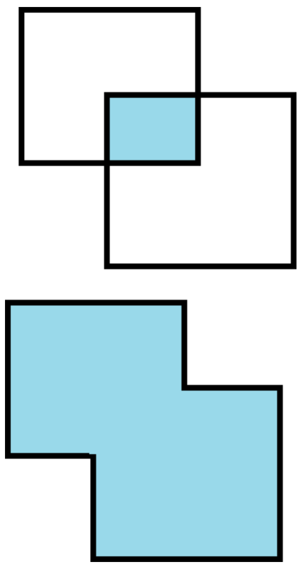


Fig 4.5 IoU

Average Precision (AP)

Average Precision (AP) is calculated as the area under a precision vs. recall curve fora set of predictions.

Recall is calculated as the ratio of the total predictions made by the model under a class with a total of existing labels for the class. Precision refers to the ratio of true positives with respect to the total predictions made by the model.

Recall and precision offer a trade-off that is graphically represented into a curve by varying the classification threshold. The area under this precision vs. recall curve gives us the Average Precision per class for the model. The average of this value, taken over all classes, is called mean Average Precision (mAP).

In object detection, precision and recall aren't used for class predictions. Instead, they serve as predictions of boundary boxes for measuring the decision performance. An IoU value > 0.5. is taken as a positive prediction, while an IoU value < 0.5 is a negative prediction.

What is YOLO?

You Only Look Once (YOLO) proposes using an end-to-end neural network that makes predictions of bounding boxes and class probabilities all at once. It differs from the approach taken by previous object detection algorithms, which repurposed classifiers to perform detection.

Following a fundamentally different approach to object detection, YOLO achieved state-of-the-art results, beating other real-time object detection algorithms by a large margin.

While algorithms like Faster RCNN work by detecting possible regions of interest using the Region Proposal Network and then performing recognition on those regions separately, YOLO performs all of its predictions with the help of a single fully connected layer.

Methods that use Region Proposal Networks perform multiple iterations for the same image, while YOLO gets away with a single iteration.

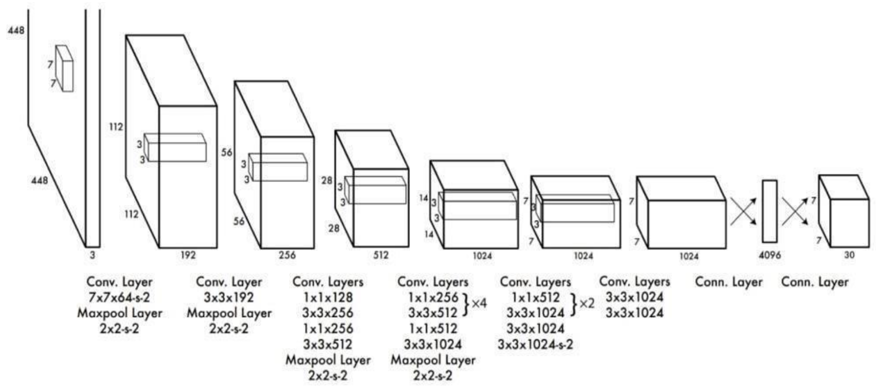

Conv. Layer 7x7x64-s-2 Maxpool Layer 2x2-s-2
Conv. Layer 3x3x192 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x128 3x3x256 1x1x256 3x3x512 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x256 3x3x512 ×4 1x1x512 3x3x1024 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x512 3x3x1024 ×2 3x3x1024 3x3x1024-s-2
Conv. Layers 3x3x1024 3x3x1024
Conn. Layer
Conn. Layer
448
448
3
112
112
192
56
56
256
28
28
512
14
14
1024
7
7
1024
1024
4096
30

Fig 4.6 Architecture of YOLO

Several new versions of the same model have been proposed since the initial release of YOLO in 2015, each building on and improving its predecessor. Here's a timeline showcasing YOLO's development in recent years.

How does YOLO work? YOLO Architecture

The YOLO algorithm takes an image as input and then uses a simple deep convolutional neural network to detect objects in the image. The architecture of the CNN model that forms the backbone of YOLO is shown below.

The first 20 convolution layers of the model are pre-trained using ImageNet by plugging in a temporary average pooling and fully connected layer. Then, this pre-trained model is converted to perform detection since previous research showcased

that adding convolution and connected layers to a pre-trained network improves performance. YOLO's final fully connected layer predicts both class probabilities and bounding box coordinates.

YOLO divides an input image into an S × S grid. If the center of an object falls into a grid cell, that grid cell is responsible for detecting that object. Each grid cell predicts B bounding boxes and confidence scores for those boxes. These confidence scores reflect how confident the model is that the box contains an object and how accurate it thinks the predicted box is.

YOLO predicts multiple bounding boxes per grid cell. At training time, we only want one bounding box predictor to be responsible for each object. YOLO assigns one predictor to be "responsible" for predicting an object based on which prediction has the highest current IOU with the ground truth. This leads to specialization between the bounding box predictors. Each predictor gets better at forecasting certain sizes, aspect ratios, or classes of objects, improving the overall recall score.

One key technique used in the YOLO models is **non-maximum suppression (NMS).** NMS is a post-processing step that is used to improve the accuracy and efficiency of object detection. In object detection, it is common for multiple bounding boxes to be generated for a single object in an image. These bounding boxes may overlap or be located at different positions, but they all represent the same object. NMS is used to identify and remove redundant or incorrect bounding boxes and to output a single bounding box for each object in the image.

Now, let us look into the improvements that the later versions of YOLO have brought to the parent model.

YOLO v2

YOLO v2, also known as YOLO9000, was introduced in 2016 as an improvement over the original YOLO algorithm. It was designed to be faster and more accurate than YOLO and to be able to detect a wider range of object classes. This updated version also uses a different CNN backbone called Darknet-19, a variant of the VGGNet architecture with simple progressive convolution and pooling layers.

One of the main improvements in YOLO v2 is the use of anchor boxes. Anchor boxes are a set of predefined bounding boxes of different aspect ratios and scales. When

predicting bounding boxes, YOLO v2 uses a combination of the anchor boxes and the predicted offsets to determine the final bounding box. This allows the algorithm to handle a wider range of object sizes and aspect ratios.

Another improvement in YOLO v2 is the use of batch normalization, which helps to improve the accuracy and stability of the model. YOLO v2 also uses a multi-scale training strategy, which involves training the model on images at multiple scales and then averaging the predictions. This helps to improve the detection performance of small objects.

YOLO v2 also introduces a new loss function better suited to object detection tasks. The loss function is based on the sum of the squared errors between the predicted and ground truth bounding boxes and class probabilities.

The results obtained by YOLO v2 compared to the original version and other contemporary models are shown below.

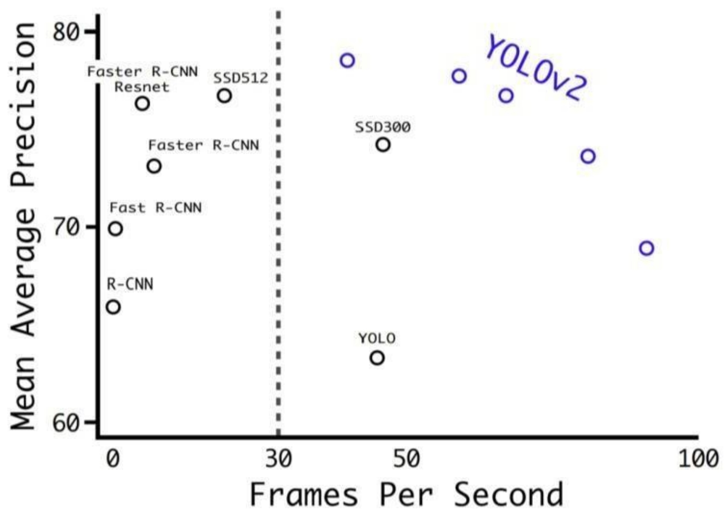


Fig 4.7 Comparison of the results obtained by YOLO v2

YOLO v3

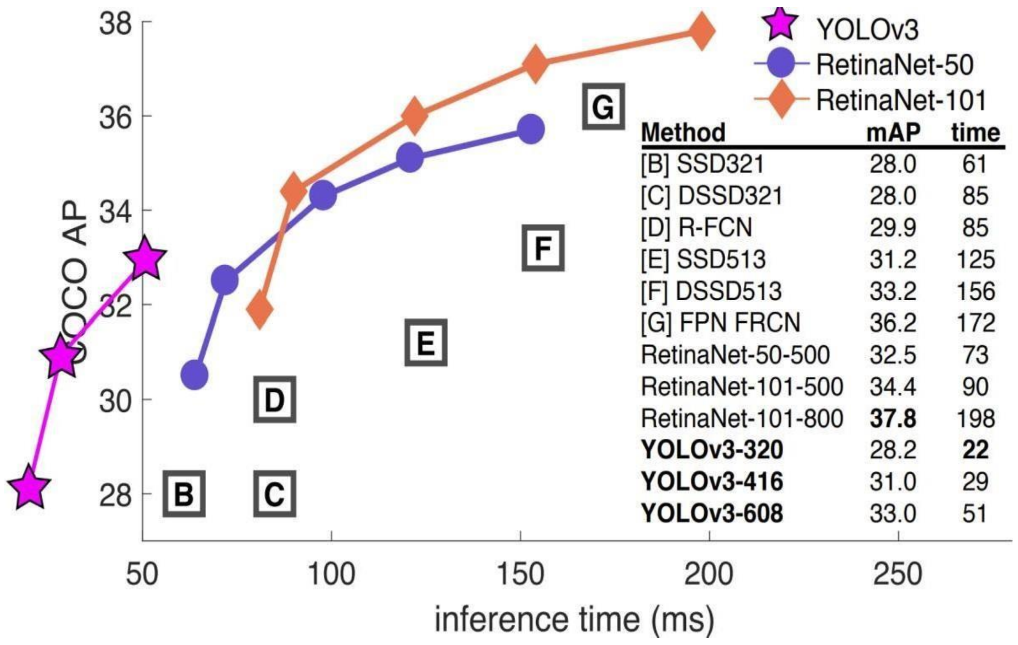
YOLO v3 is the third version of the YOLO object detection algorithm. It was introduced in 2018 as an improvement over YOLO v2, aiming to increase the accuracy and speed of the algorithm.

One of the main improvements in YOLO v3 is the use of a new CNN architecture called Darknet-53. Darknet-53 is a variant of the ResNet architecture and is designed specifically for object detection tasks. It has 53 convolutional layers and is able to achieve state-of-the-art results on various object detection benchmarks.

Another improvement in YOLO v3 are anchor boxes with different scales and aspect ratios. In YOLO v2, the anchor boxes were all the same size, which limited the ability of the algorithm to detect objects of different sizes and shapes. In YOLO v3 the anchor boxes are scaled, and aspect ratios are varied to better match the size and shape of the objects being detected.

YOLO v3 also introduces the concept of "feature pyramid networks" (FPN). FPNs are a CNN architecture used to detect objects at multiple scales. They construct a pyramid of feature maps, with each level of the pyramid being used to detect objects at a different scale. This helps to improve the detection performance on small objects, as the model is able to see the objects at multiple scales.

In addition to these improvements, YOLO v3 can handle a wider range of object sizes and aspect ratios. It is also more accurate and stable than the previous versions of YOLO.



Method	mAP	time
[B] SSD321	28.0	61
[C] DSSD321	28.0	85
[D] R-FCN	29.9	85
[E] SSD513	31.2	125
[F] DSSD513	33.2	156
[G] FPN FRCN	36.2	172
RetinaNet-50-500	32.5	73
RetinaNet-101-500	34.4	90
RetinaNet-101-800	**37.8**	198
YOLOv3-320	28.2	**22**
YOLOv3-416	31.0	29
YOLOv3-608	33.0	51

Fig 4.8 Comparison of the results obtained by YOLO v3

YOLO v4

Note: Joseph Redmond, the original creator of YOLO, has left the AI community a few years before, so YOLOv4 and other versions past that are not his official work. Some of them are maintained by co-authors, but none of the releases past YOLOv3 is considered the "official" YOLO.

YOLO v4 is the fourth version of the YOLO object detection algorithm introduced in 2020 by Bochkovskiy et al. as an improvement over YOLO v3.

The primary improvement in YOLO v4 over YOLO v3 is the use of a new CNN architecture called CSPNet (shown below). CSPNet stands for "Cross Stage Partial Network" and is a variant of the ResNet architecture designed specifically for object detection tasks. It has a relatively shallow structure, with only 54 convolutional layers. However, it can achieve state-of-the-art results on various object detection benchmarks.

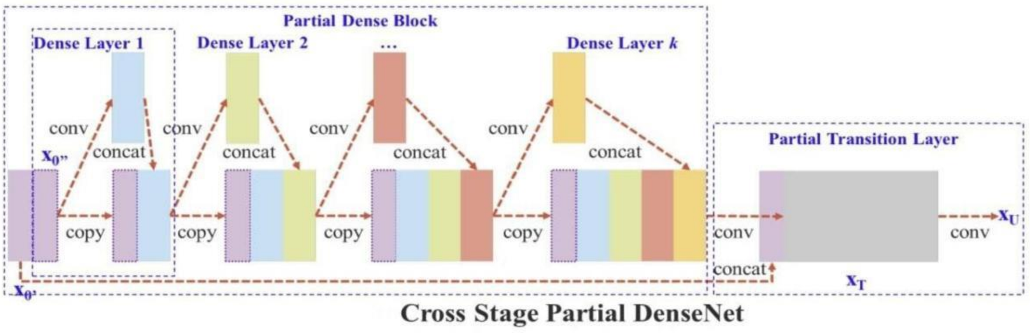


Fig 4.9 Architecture of CSPNet

Both YOLO v3 and YOLO v4 use anchor boxes with different scales and aspect ratios to better match the size and shape of the detected objects. YOLO v4 introduces a new method for generating the anchor boxes, called "k-means clustering." It involves using a clustering algorithm to group the ground truth bounding boxes into clusters and then using the centroids of the clusters as the anchor boxes. This allows the anchor boxes to be more closely aligned with the detected objects' size and shape.

While both YOLO v3 and YOLO v4 use a similar loss function for training the model, YOLO v4 introduces a new term called "GHM loss." It's a variant of the focal loss function and is designed to improve the model's performance on imbalanced datasets.

YOLO v4 also improves the architecture of the FPNs used in YOLO v3.

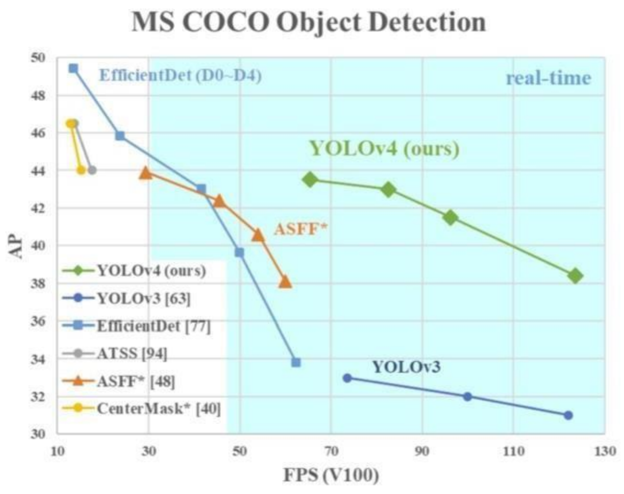


Fig 4.10 Comparative performance of YOLO v4

YOLO v5

YOLO v5 was introduced in 2020 by the same team that developed the original YOLO algorithm as an open-source project and is maintained by Ultralytics. YOLO v5 builds upon the success of previous versions and adds several new features and improvements.

Unlike YOLO, YOLO v5 uses a more complex architecture called EfficientDet (architecture shown below), based on the EfficientNet network architecture. Using a more complex architecture in YOLO v5 allows it to achieve higher accuracy and better generalization to a wider range of object categories.

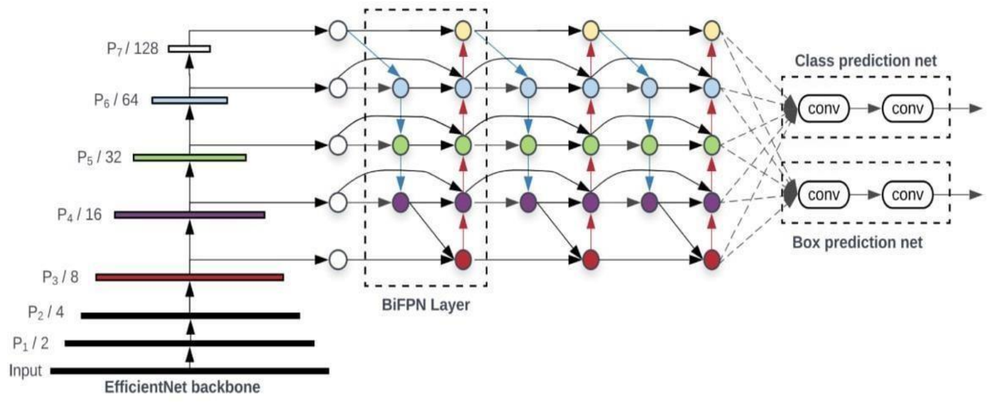


Fig 4.11 Architecture of YOLO v5

Another difference between YOLO and YOLO v5 is the training data used to learn the object detection model. YOLO was trained on the PASCAL VOC dataset, which consists of 20 object categories. YOLO v5, on the other hand, was trained on a larger and more diverse dataset called D5, which includes a total of 600 object categories.

YOLO v5 uses a new method for generating the anchor boxes, called "dynamic anchor boxes." It involves using a clustering algorithm to group the ground truth bounding boxes into clusters and then using the centroids of the clusters as the anchor boxes. This allows the anchor boxes to be more closely aligned with the detected objects' size and shape.

YOLO v5 also introduces the concept of "spatial pyramid pooling" (SPP), a type of pooling layer used to reduce the spatial resolution of the feature maps. SPP is used to improve the detection performance on small objects, as it allows the model to see the objects at multiple scales. YOLO v4 also uses SPP, but YOLO v5 includes several improvements to the SPP architecture that allow it to achieve better results.

YOLO v4 and YOLO v5 use a similar loss function to train the model. However, YOLO v5 introduces a new term called "CIoU loss," which is a variant of the IoU loss function designed to improve the model's performance on imbalanced dataset.

CHAPTER – 5
FEASIBILITY STUDY

The feasibility of the project is analyzed in this phase and business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis the feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the company. For feasibility analysis, some understanding of the major requirements for the system is essential.

Three key considerations involved in the feasibility analysis are

- TECHNICAL FEASIBILITY
- SOCIAL FEASIBILITY
- ECONOMICAL FEASIBILITY

5.1 TECHNICAL FEASIBILITY

This study is carried out to check the technical feasibility, that is, the technical requirements of the system. Any system developed must not have a high demand on the available technical resources. This will lead to high demands on the available technical resources. This will lead to high demands being placed on the client. Th developed system must have a modest requirement, as only minimal or null changes are required for implementing this system.

5.2 SOCIAL FEASIBILITY

The aspect of study is to check the level of acceptance of the system by the user. This includes the process of training the user to use the system efficiently. The user must not feel threatened by the system, instead must accept it as a necessity. The level of acceptance by the users solely depends on the methods that are employed to educate the user about the system and to make him familiar with it. His level of confidence must be raised so that he is also able to make some constructive criticism, which is welcomed, as he is the final user of the system.

5.3 ECONOMICAL FEASIBILITY

This study is carried out to check the economic impact that the system will have on the organization. The amount of fund that the company can pour into the research and development of the system is limited. The expenditures must be justified. Thus the developed system as well within the budget and this was achieved because most of the technologies used are freely available. Only the customized products had to be purchased.

CHAPTER - 6
SYSTEM ANALYSIS

6.1 EXISTING SYSTEM

The one of the method available for the vehicle detection is by using R-CNN. This is known as Region based CNN, it uses the selective search algorithm that generates approximately 2000 region proposals and features are extracted. These features are then passed in an SVM model to classify the object present in the region proposal. Finally, Bounding box regressor is used to precisely locate the bounding box in the image.

The second popular method available is by directly using YOLO v4 which is the previous version of the algorithm that we are currently using. This model divides an incoming image into numerous grids and calculate the probability of an object resides inside that grid. The algorithm groups nearby high-value probability grids as asingle object. It uses C as a primary language.

6.1.1 Drawbacks of Existing System

The collective drawbacks of these systems are they take a lot of time and space and also need more computation power especially YOLO v4, it uses C language.

6.2 PROPOSED SYSTEM

In this project, we use YOLO v5 integrated with multiple convolutional layers and the dataset that is used for training and testing is standard MOT dataset which is diverse and covers all types of environments. Google colab is used as a platform for this system.

6.2.1 Advantages of Proposed System

YOLO v5 features are an added advantage but that’s not the whole part, Here the algorithm integrates with the Multi layer feature fusion CNN map to the pre trained yolo algorithm which creates the entire difference. This will increase the accuracy tremendously and also decrease the time taking for prediction as well as training. So, It is expected to outperform the existing system.

CHAPTER - 7
SYSTEM DESIGN

System design is the process of defining the architecture, components, modules, interfaces, and data for a system to satisfy specified requirements. One couldsee it as the application of systems theory to product development. There is some overlap with the disciplines of systems analysis, systems architecture and systems engineering. If the broader topic of product development "blends the perspective of marketing, design, and manufacturing into a single approach to product development, then design is the act of taking the marketing information and creating the design of theproduct to be manufactured.

System design is therefore the process of defining and developing systems to satisfy specified requirements of the user.

7.1 PROPOSED ARCHITECTURE

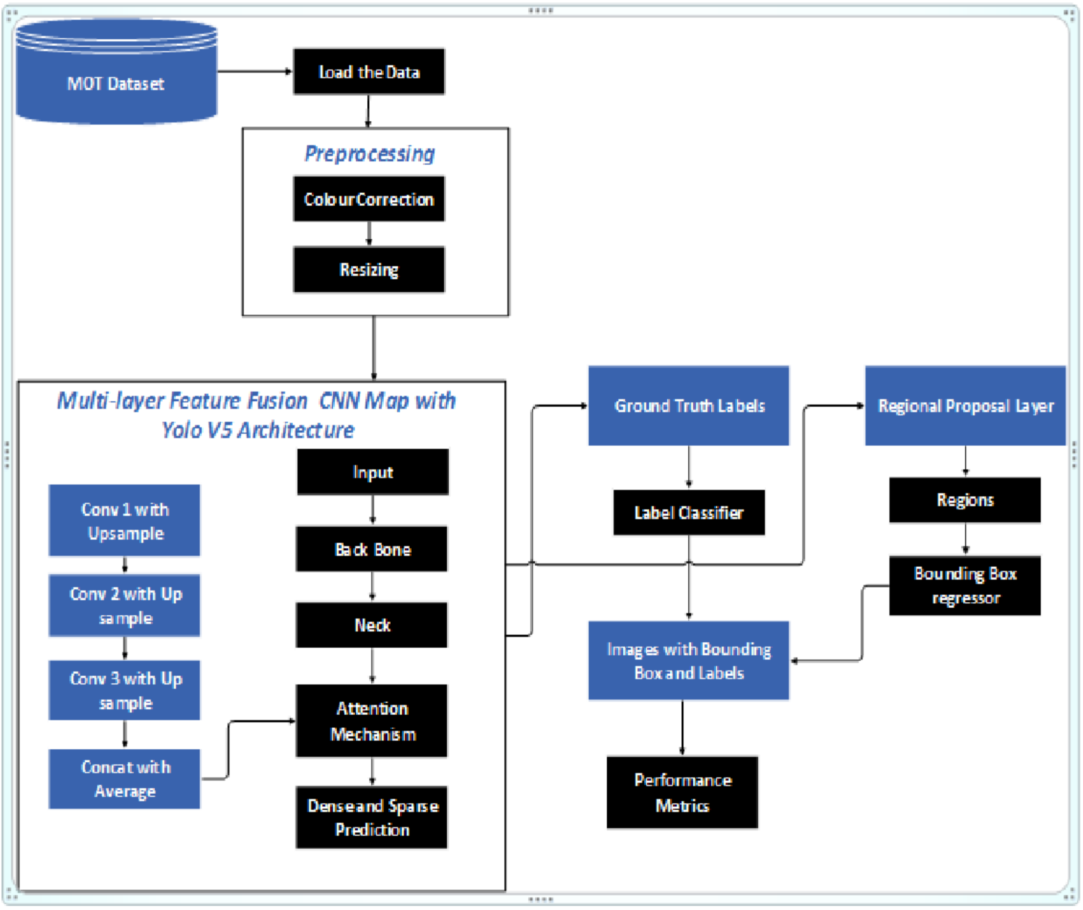


Fig 7.1 Architecture of proposed system

7.2 UML DIAGRAMS

The Unified Modeling Language (UML) is a standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems. The UML represents a collection of best engineering practices that have proven successful in the modelling of large and complex systems. The UML is a very important part of developing objects-oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects. Using the UML helps project teams communicate, explore potential designs, and validate the architectural design of the software.

As the strategic value of software increases for many companies, the industry looks for techniques to automate the production of software and to improve quality and reduce cost and time-to-market. These techniques include component technology, visual programming, patterns and frameworks. Businesses also seek techniques to manage the complexity of systems as they increase in scope and scale. In particular, they recognize the need to solve recurring architectural problems, such as physical distribution, concurrency, replication, security, load balancing and fault tolerance. Additionally, the development for the World Wide Web, while making some things

simpler, has exacerbated these architectural problems. The Unified Modeling Language (UML) was designed to respond to these needs. Simply, Systems design refers to the process of defining the architecture, components, modules, interfaces, and data for a system to satisfy specified requirements which can be done easily through UML diagrams.

Contents of UML

In general, a UML diagram consists of the following features:

- **Entities:** These may be classes, objects, users or systems behaviors.
- Relationship Lines that model the relationships between entities in the system.
- **Generalization --** a solid line with an arrow that points to a higher abstraction of the present item.
- **Association --** a solid line that represents that one entity uses another entity as part of its behaviour.

- **Dependency --** a dotted line with an arrowhead that shows one entity depends on the behaviour of another entity.

In this project four basic UML diagrams have been explained

1) Class Diagram
2) Use Case Diagram
3) Sequence Diagram
4) Activity Diagram
5) Deployment Diagram
6) Component Diagram

7.2.1 Class Diagram

UML class diagrams model static class relationships that represent the fundamental architecture of the system. Note that these diagrams describe the relationships between classes, not those between specific objects instantiated from those classes. Thus the diagram applies to all the objects in the system.

A class diagram consists of the following features:

- **Classes:** These titled boxes represent the classes in the system and contain information about the name of the class, fields, methods and access specifies. Abstract roles of the Class in the system can also be indicated.
- **Interfaces:** These titled boxes represent interfaces in the system and contain information about the name of the interface and its methods. Relationship Lines that model the relationships between classes and interfaces in the system.
- **Dependency:** A dotted line with an open arrowhead that shows one entity depends on the behavior of another entity. Typical usages are to represent that one class instantiates another or that it uses the other as an input parameter
- **Aggregation:** Represented by an association line with a hollow diamond at the tail end. An aggregation models the notion that one object uses another object without "owning" it and thus is not responsible for its creation or destruction.
- **Inheritance:** A solid line with a solid arrowhead that points from a sub-class to a super class or from a sub-interface to its super-interface.
- **Implementation:** A dotted line with a solid arrowhead that points from a class to the interface that it implement

- **Composition:** Represented by an association line with a solid diamond at the tail end. A composition models the notion of one object "owning" another and thus being responsible for the creation and destruction of another object.

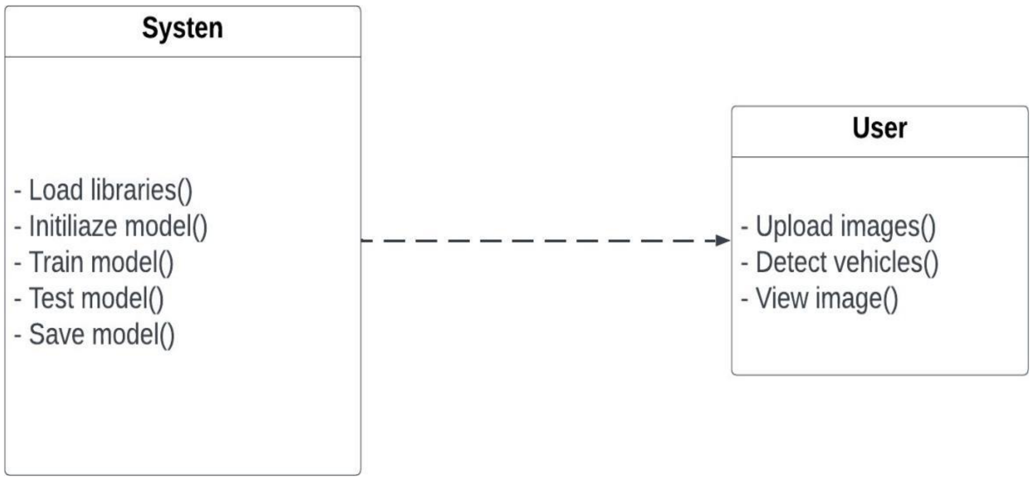


Fig 7.2.1 Class Diagram

7.2.2 Use case diagram

A use case diagram in the Unified Modeling Language (UML) is a type of behavioural diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms.

A use case is a methodology used in system analysis to identify, clarify, and organize system requirements. The use case is made up of a set of possible sequences of interactions between systems and users in a particular environment and related to a particular goal. It consists of a group of elements (for example, classes and interfaces) that can be used together in a way that will have an effect larger than the sum of the separate elements combined. The use case should contain all system activities that have significance to the users. A use case can be thought of as a collection of possible scenarios related to a particular goal, indeed, the use case and goal are sometimes considered to be synonymous.

The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.

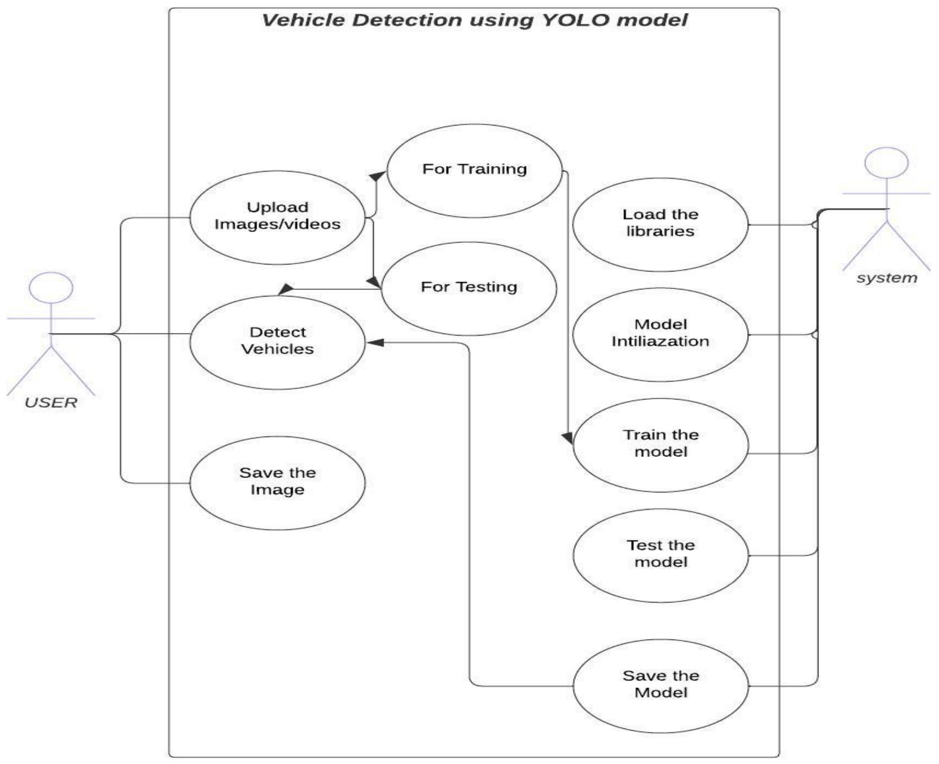


Fig 7.2.2 Use Case Diagram

- **Parts of Use cases**

A use case describes a sequence of actions that provide something of measurable value to an actor and is drawn as a horizontal ellipse.

- **Actors**

An actor is a person, organization, or external system that plays a role in one or more interactions with the system.

- **System boundary boxes (optional)**

A rectangle is drawn around the use cases, called the system boundary box, to indicate the scope of system. Anything within the box represents functionality that is in scope and anything outside the box is not **Relationships.**

- **Include**

In one form of interaction, a given use case may include another. "Include is a Directed Relationship between two use cases, implying that the behavior of the included use case is inserted into the behavior of the including use case".

The first use case often depends on the outcome of the included use case. This is useful for extracting truly common behaviours from multiple use cases into a single description. The notation is a dashed arrow from the including to the included use case, with the label "«include»". This usage resembles a macro expansion where the included use case behavior is placed inline in the base use case behavior. There are no parametersor return values. To specify the location in a flow of events in which the base use case includes the behavior of another, you simply write include followed by the name of usecase you want to include, as in the following flow for track order.

- **Extend**

In another form of interaction, a given use case (the extension) may extend another. This relationship indicates that the behavior of the extension use case may be inserted in the extended use case under some conditions. The notation is a dashed arrow from the extension to the extended use case, with the label "«extend»". The notes or constraints may be associated with this relationship to illustrate the conditions under which this behavior will be executed. Modelers use the «extend» relationship to indicate use cases that are "optional" to the base use case. Depending on the modeler's approach "optional" may mean "potentially not executed with the base use case" or it may mean "not required to achieve the base use case goal".

- **Generalization**

In the third form of relationship among use cases, a generalization/ specialization relationship exists. A given use case may have common behaviours, requirements, constraints, and assumptions with a more general use case. In this case, describe them once, and deal with it in the same way, describing any differences in

the specialized cases. The notation is a solid line ending in a hollow triangle drawn from the specialized to the more general use case (following the standard generalization notation).

- **Associations**

Associations between actors and use cases are indicated in use case diagrams by solid lines. An association exists whenever an actor is involved with an interaction described by a use case. Associations are modeled as lines connecting use cases and actors to one another, with an optional arrowhead on one end of the line. The arrowhead is often used to indicate the direction of the initial invocation of the relationship or to indicate the primary actor within the use case. The arrowheads imply control flow and should not be confused with data flow.

- **STEPS TO DRAW USE CASES**
  - Identifying Actor
  - Identifying Use cases
  - Review your use case for completeness

7.2.3 Sequence Diagram

A sequence diagram in Unified Modeling Language (UML) is a kind ofinteraction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. A Sequence diagram depicts thesequence of actions that occur in a system. The invocation of methods in each object,and the order in which the invocation occurs is captured in a Sequence diagram. This makes the Sequence diagram a very useful tool to easily represent the dynamic behaviour of a system.

- **Elements of sequence diagram**

The sequence diagram is an element that is used primarily to showcase the interaction that occurs between multiple objects. This interaction will be shown over certain period of time. Because of this, the first symbol that is used is one that symbolizes the object.

- **Lifeline**

A lifeline will generally be generated, and it is a dashed line that sits vertically, and the top will be in the form of a rectangle. This rectangle is used to indicate both the instance and the class. If the lifeline must be used to denote an object, it will be underlined.

- **Messages**

To showcase an interaction, messages will be used. These messages will come in the form of horizontal arrows, and the messages should be written on top of the arrows. If the arrow has a full head, and it’s solid, it will be called a synchronous call.

If the solid arrow has a stick head, it will be an asynchronous call. Stick heads with dash arrows are used to represent return messages.

- **Objects**

  Objects will also be given the ability to call methods upon themselves, and they can add net activation boxes. Because of this, they can communicate with others to show multiple levels of processing. Whenever an object is eradicated or erased from memory, the "X" will be drawn at the lifeline's top, and the dash line will not be drawn beneath it. This will often occur as a result of a message. If a message is sent from the outside of the diagram, it can be used to define a message that comes from a circle that is filled in. Within a UML based model, a Super step is a collection of steps which result from outside stimuli.

Steps to Create a Sequence Diagram

- Set the context for the interaction, whether it is a system, subsystem, operation or class.
- Set the stage for the interaction by identifying which objects play a role in interaction.
- Set the lifetime for each object.
- Start with the message that initiates the interaction.
- Visualize the nesting of messages or the points in time during actual computation.
- Specify time and space constraints, adorn each message with timing mark and attach suitable time or space constraints.
- Specify the flow of control more formally, attach pre and post conditions to each message.

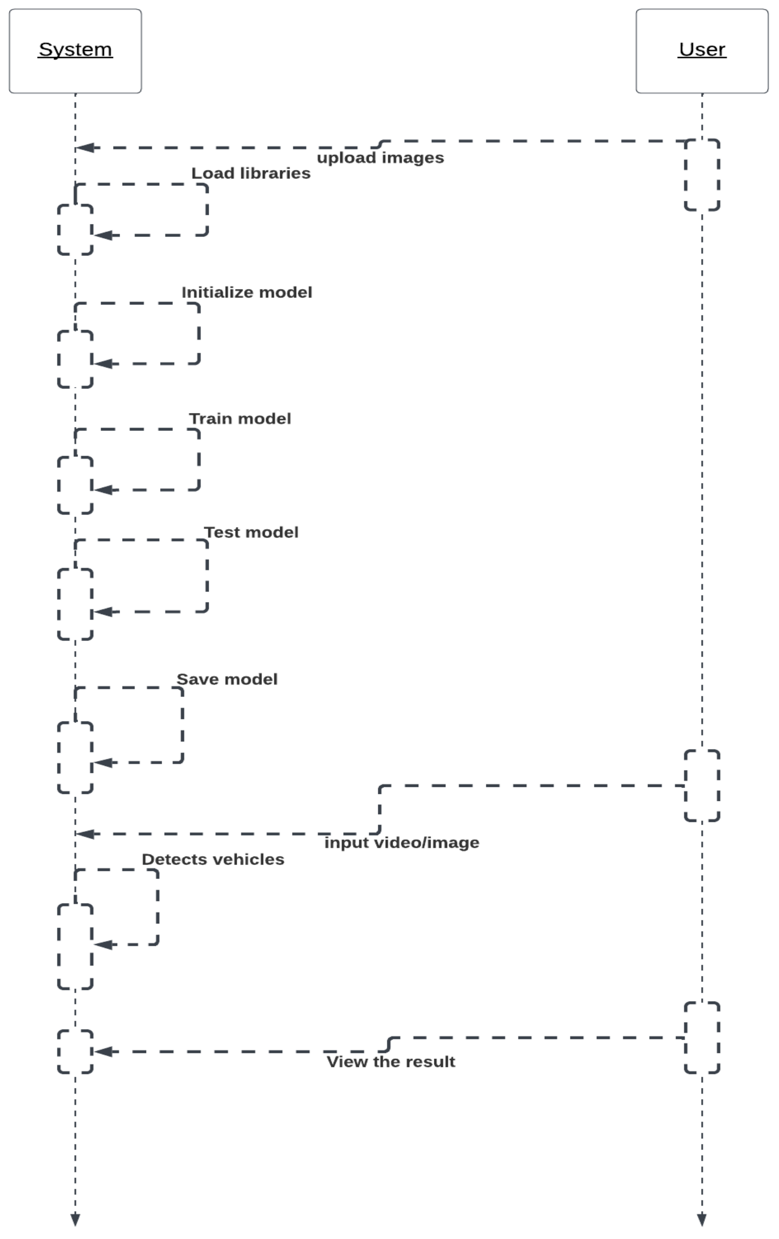


Fig 7.2.3 Sequence Diagram

7.2.4 Activity Diagram

Activity diagram is another important diagram in UML to describe dynamic aspects of the system. Activity diagram is basically a flow chart to represent the flow form one activity to another activity. The activity can be described as an operation of the system. So the control flow is drawn from one operation to another. This flow can be sequential, branched or concurrent. Activity diagrams deals with all type of flow control by using different elements like fork, join etc.

How to draw Activity Diagram?

Activity diagrams are mainly used as a flow chart consists of activities performed by the system. But activity diagram are not exactly a flow chart as they have some additional capabilities. These additional capabilities include branching, parallel flow, swim lane etc. Before drawing an activity diagram we must have a clear understanding about the elements used in activity diagram. The main element of an activity diagram is the activity itself. An activity is a function performed by the system. After identifying the activities we need to understand how they are associated with constraints and conditions. So before drawing an activity diagram we should identify the following elements.

- Activities
- Association
- Conditions
- Constraints

The following are the basic notational elements that can be used to make up a diagram:

Initial state

An initial state represents a default vertex that is the source for a single transition to the default state of a composite state. There can be at most one initial vertex in a region. The outgoing transition from the initial vertex may have a behavior, but not a trigger or guard. It is represented by Filled circle, pointing to the initial state.

Final state

A special kind of state signifying that the enclosing region is completed. If the enclosing region is directly contained in a state machine and all other regions in the state machine also are completed, then it means that the entire state machine is completed. It is represented by Hollow circle containing a smaller filled circle, indicating the final state.

Rounded rectangle

It denotes a state. Top of the rectangle contains a name of the state. Can contain a horizontal line in the middle, below which the activities that are done in that state are indicated.

Arrow

It denotes transition. The name of the event (if any) causing this transition labels the arrow body.

Steps To Construct Activity Diagram

- Identify the preconditions of the workflow
- Collect the abstractions that are involved in the operations
- Beginning at the operation's initial state, specifythe activities and actions.
- Use branching to specify conditional paths and iterations
- Use forking & joining to specify parallel flows of control.

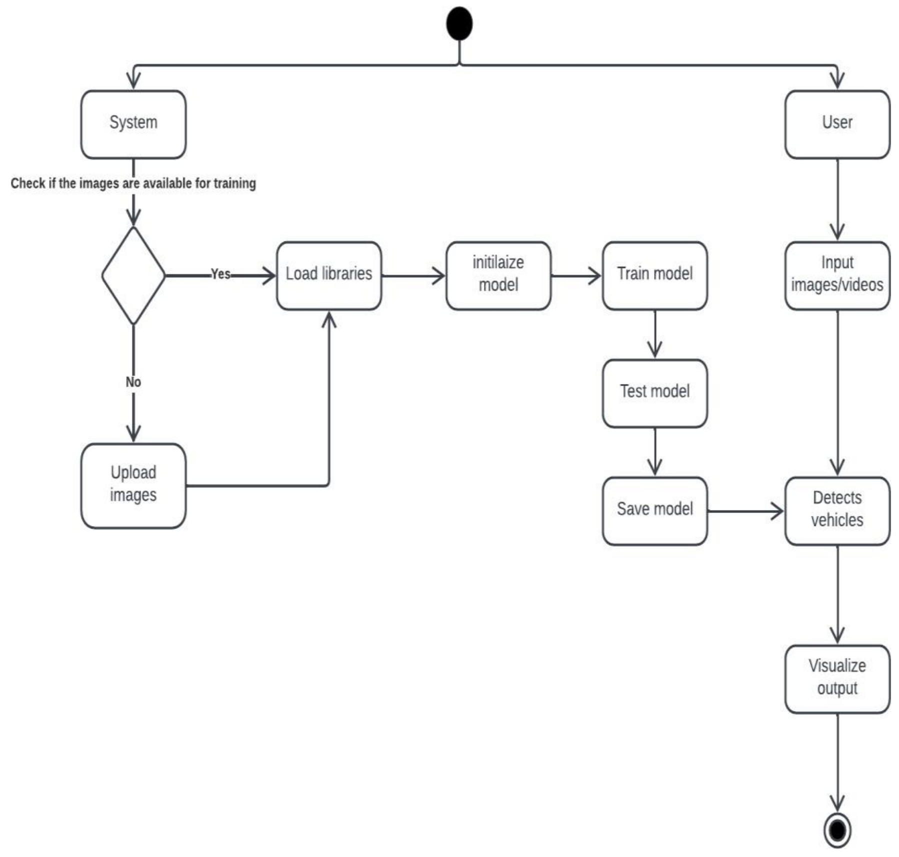


Fig 7.2.4 Activity Diagram

7.2.5 Deployment Diagram

Deployment diagrams are used to visualize the topology of the physical components of a system, where the software components are deployed.

Deployment diagrams are used to describe the static deployment view of a system. Deployment diagrams consist of nodes and their relationships.

Purpose of Deployment Diagrams

The term Deployment itself describes the purpose of the diagram. Deployment diagrams are used for describing the hardware components, where software components are deployed. Component diagrams and deployment diagrams are closely related.

Component diagrams are used to describe the components and deployment diagrams shows how they are deployed in hardware.

UML is mainly designed to focus on the software artifacts of a system. However, these two diagrams are special diagrams used to focus on software and hardware components.

Most of the UML diagrams are used to handle logical components but deployment diagrams are made to focus on the hardware topology of a system. Deployment diagrams are used by the system engineers.

The purpose of deployment diagrams can be described as −

- Visualize the hardware topology of a system.
- Describe the hardware components used to deploy software components.
- Describe the runtime processing nodes.

Steps to Draw a Deployment Diagram?

Deployment diagram represents the deployment view of a system. It is related to the component diagram because the components are deployed using the deployment diagrams. A deployment diagram consists of nodes. Nodes are nothing but physical hardware used to deploy the application.

Deployment diagrams are useful for system engineers. An efficient deployment diagram is very important as it controls the following parameters −

- Performance
- Scalability
- Maintainability
- Portability

Before drawing a deployment diagram, the following artifacts should be identified −

- Nodes
- Relationships among nodes

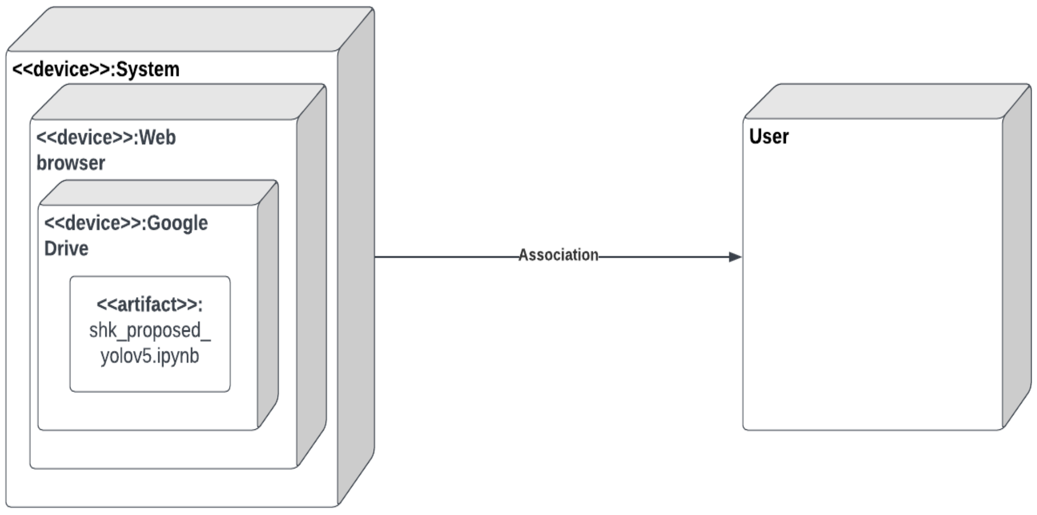


Fig 7.2.5 Deployment diagram

7.2.6 Component Diagram

Deployment diagrams are used to visualize the topology of the physical components of a system, where the software components are deployed. Deployment diagrams are used to describe the static deployment view of a system. Deployment diagrams consist of nodes and their relationships.

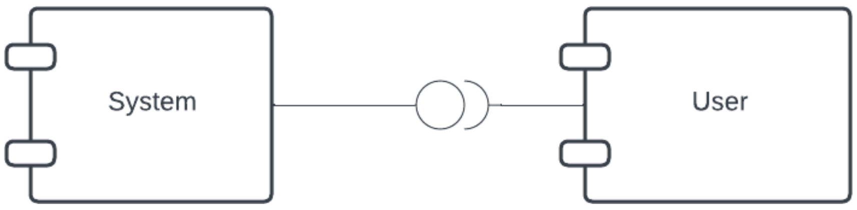


Fig 7.2.6 Component Diagram

CHAPTER - 8
SYSTEM IMPLEMENTATION

8.1 IMPLEMENTATION PROCESS

Implementation is the stage of the project when the theoretical design is turnedout into a working system. Thus, it can be considered to be the most critical stage in achieving a successful new system and in giving the user, confidence that the new system will work and be effective.

The implementation stage involves careful planning, investigation of the existing system and its constraints on implementation, designing of methods to achieve changeover and evaluation of changeover methods.

8.2 MODULES

1. Preprocessing
2. Multi layer feature fusion CNN map with Yolo v5 Architecture
3. Label Classifier and Bounding Box regressor
4. Performance Metrics

Module Description

1. **Preprocessing**
   - The data is loaded into the model. This data contains a lot of images for training, validation and testing respectively.
   - The main thing that happens in this phase is the color and size of the images is altered according to the rules of Yolo.
   - The images must be 416 x 416 pixels.
2. **Multi layer feature fusion CNN map with Yolo v5 Architecture**
   - This is the heart of the system. This is where all the training and learning is done.

- A Convolutional Neural Network (CNN) is a type of deep learning algorithm that is particularly well-suited for image recognition and processing tasks. It is made up of multiple layers, including convolutional layers, pooling layers, and fully connected layers.

- Here, the image undergoes multiple CNN layers (3 to be precise) and the concatenated result is given as an input for the Attention mechanism of Yolo.

- Yolo extracts the prediction model by combining the result given and processing it again in its own way.

3. **Label Classifier and Bounding Box regressor**

- From the prediction model, label classifier and bounding box regressor are formed.

- Label classifier will classify the detected object whether it is a car or truck etc.

- Bounding Box regressor will draw the box around the object and the name given by the label classifier.

4. **Performance Metrics**

- This is the final module where the detected object is given a confidence score stating how confident the system is towards its prediction.

- Higher the confidence, higher the object is classified accurately.

CHAPTER - 9
TESTING

A “program unit” stands for a routine or a collection of routines implemented by an individual programmer. It might even be a stand-alone program or a functional unit a larger program.

9.1 UNIT TESTING

Unit testing is performed prior to integration of the unit into a larger system. It is like coding and debugging ->unit testing ->integration testing. A program unit mustbe tested for functional tests, performance tests, stress tests and structure tests.

Functional tests refer to executing the code with standard inputs, for which the results will occur within the expected boundaries. Performance test determines the execution time spent in various parts of the unit, response time, device utilization and throughput. Performance testing will help the tuning of the system.

Stress tests drive the system to its limits. They are designed to internationally break the unit. Structure tests verify logical execution along different execution paths. Functional, performance and stress tests are collectively known as “black box testing”. Structure testing is referred to as “white box” or “glass box” testing. Program errors can be classified as missing path errors, computational errors and domain errors.

Even if it looks like all possible execution paths have been tested, there might still exist some more paths. A missing path error occurs, when a branching statement and the associated computations are accidentally omitted. Missing paths can be detected only by functional specifications. A domain error occurs when a program traverses the wrong path because of an incorrect predicate in a branching statement. When a test case fails to detect a computational error there is said to be a coincidental error.

Debugging

Debugging is eliminating the cause of known errors. Commonly used debugging techniques are induction, deduction and backtracking. Debugging by induction involves the following steps:

- Collect all the information about test details and test results
- Look for patterns
- Form one or more hypotheses and rank/classify them.
- Prove/disprove hypotheses. Re examine
- Implement appropriate corrections
- Verify the corrections. Re run the system and test again until satisfactory
- Debugging by deduction involves the following steps:
- List possible causes for observed failure.
- Use the available information to eliminate various hypotheses.
- Prove/disprove the remaining hypotheses.
- Determine the appropriate correction.
- Carryout the corrections and verify.

Debugging by backtracking involves working backward in the source code from point where the error was observed. Run additional test cause and collect more information.

9.2 INTEGRATION TESTING

Integration testing strategies include bottom-up (traditional), top-down and sandwich strategies. Bottom-up integration consists of unit testing, followed by testing entire system. Unit testing tries to discover errors in modules. Modules are tested independently in an artificial environment known as “test harness”. Test harnesses provide data environments and calling sequences for the routines and subsystem that are being tested in isolation.

Disadvantages of bottom-up testing include that harness preparation, which can sometimes take about 50% or more of the coding and debugging effort for a smaller product. After testing all the modules independently and in isolation, they are linked and executed in one single integration run. This is known as “Big bang” approach to integration testing. Isolating sources of errors is difficult in “big bang” approach.

Top-down integration starts with main routine and one or two immediately next lower level routines. After a through checking the top level becomes a test harness to its immediate subordinate routines. Top-down integration offers the following advantages.
System integration is distributed throughout the implementation phase. Modules are integrated as theyare developed.

- Top-level interfaces are first test.
- Top-level routines provide a naturaltest harness for lower-level routines.
- Errors are localized to the new modules and interfaces that are being added.

Though top-down integrations seem to offer better advantages, it may not be applicable in certain situations. Sometimes it may be necessary to test certain low- level modules first. In such situations, a sandwich strategy is preferable. Sandwich integration is mostly top-down, but bottom-up techniques are used on some modules and sub systems. This mixed approach retains the advantages of both strategies.

9.3 SYSTEM TESTING

System testing involves two activities: Integration testing and Acceptance testing. Integration strategy stresses on the order in which modules are written, debugged and unit tested. Acceptance test involves functional tests, performance tests and stress tests to verify requirements fulfillment. System checking checks the interface, decision logic, control flow, recovery procedures and throughput, capacity and timing characteristics of the entire system.

9.4 ACCEPTANCE TESTING

Acceptance testing involves planning and execution of functional tests, performance tests and stress tests in order to check whether the system implemented satisfies the requirements specifications. Quality assurance people as well as customers may simultaneously develop acceptance tests and run them. In additional to functional and performance tests, stress test are performed to determine the limits/limitations of the system developed. For example, a compiler may be tested for its symbol table overflows or a real-time system may be tested for multiple interrupts of different/same priorities.

Acceptance test tools include a test coverage analyzer, and a coding standards checker. Test coverage analyzer records the control paths followed for each test case. A timing analyzer reports the time spent in various regions of the source code under different test cases. Coding standard are stated in the product requirements. Manual inspection is usually not an adequate mechanism from detecting violations of coding standards.

TESTING OBJECTIVES

Testing is a process of execution a program with the intent of finding on errors. A good test is on that has a high probability of finding an undiscovered errors. Testing is vital to the success of the system. System testing is the state of implementation, which ensures that the system works accurately before live operations commence. System testing makes a logical assumption that the system is correct and that the system is correct and that the goals are successfully achieved.

Effective Testing Prerequisites

Integration testing

An overall test plan for the project is prepared before the start of coding.

Validation testing

This project will be tested under this testing sample data and produce the correct sample output.

Recovery testing

This project will be tested under this testing using correct data input and its product and the correct valid output without any errors.

Security testing

This project contains password to secure the data.

Test Data and Input

Taking various types of data we do the above testing. Preparation of test data plays a vital role in system testing. After preparing the test data the system under study is treated using the test data. While testing the system by using the above testing and correction methods. The system has been verified and validated by running with both.

- Run with live data
- Run with test data

Run with test data

In the case the system was run with some sample data. Specification testing wasalso done for each conditions or combinations for conditions.

Run with live data

The system was tested with the data of the old system for a particular period. Then the new reports were verified with the old one.

CHAPTER – 10
SAMPLE SOURCE CODE

YOLO requirements installation

```
!pip install -r requirements.txt
```

Coding part Of Training

```
RES_DIR = set_res_dir()
if TRAIN:
    !python train.py --data ../data.yaml --weights yolov5l.pt \
    --img 640 --epochs {EPOCHS} --batch-size 16 --name {RES_DIR}
```

Coding part for visualization and inference

```
# Function to show validation predictions saved during training.
def show_valid_results(RES_DIR):
    !ls runs/train/{RES_DIR}
    EXP_PATH = f"runs/train/{RES_DIR}"
    validation_pred_images = glob.glob(f"{EXP_PATH}/*_pred.jpg")
    print(validation_pred_images)
    for pred_image in validation_pred_images:
        image = cv2.imread(pred_image)
        plt.figure(figsize=(19, 16))
        plt.imshow(image[:, :, ::-1])
        plt.axis('off')
        plt.show()


# Helper function for inference on images.
def inference(RES_DIR, data_path):
    infer_dir_count = len(glob.glob('runs/detect/*'))
    print(f"Current number of inference detection directories: {infer_dir_count}")
    INFER_DIR = f"inference_{infer_dir_count+1}"
    print(INFER_DIR)
    !python detect.py --weights runs/train/{RES_DIR}/weights/best.pt \
    --source {data_path} --name {INFER_DIR}
    return INFER_DIR
```

```
# Visualize inference images.
def visualize(INFER_DIR):
   INFER_PATH = f"runs/detect/{INFER_DIR}"
   infer_images = glob.glob(f"{INFER_PATH}/*.jpg")
   print(infer_images)
   for pred_image in infer_images:
      image = cv2.imread(pred_image)
      plt.figure(figsize=(19, 16))
      plt.imshow(image[:, :, ::-1])
      plt.axis('off')
      plt.show()
```

Yolo algorithm

```
import argparse
import contextlib
import os
import platform
import sys
from copy import deepcopy
from pathlib import Path

FILE = Path(_file__).resolve()
ROOT = FILE.parents[1] # YOLOv5 root directory
if str(ROOT) not in sys.path:
sys.path.append(str(ROOT))  # add ROOT to PATH
if platform.system() != 'Windows':
ROOT = Path(os.path.relpath(ROOT, Path.cwd()))  # relative

from models.common import *
from models.experimental import *
from utils.autoanchor import check_anchor_order
from utils.general import LOGGER, check_version, check_yaml,
make_divisible, print_args
from utils.plots import feature_visualization
from utils.torch_utils import (fuse_conv_and_bn, initialize_weights,
model_info, profile, scale_img, select_device,time_sync)
```

```
try:
import thop  # for FLOPs computation
except ImportError:
thop = None

class Detect(nn.Module):
# YOLOv5 Detect head for detection models
stride = None  # strides computed during build
dynamic = False # force grid reconstruction
export = False # export mode

def __init_(self, nc=80, anchors=(), ch=(), inplace=True):  # detection layer
super().__init__()
self.nc = nc # number of classes
self.no = nc + 5 # number of outputs per anchor
self.nl = len(anchors) # number of detection layers
self.na = len(anchors[0]) // 2  # number of anchors
self.grid = [torch.empty(0) for _ in range(self.nl)]  # init grid
self.anchor_grid = [torch.empty(0) for _ in range(self.nl)]  # init anchor grid
self.register_buffer('anchors',
torch.tensor(anchors).float().view(self.nl, -1, 2))# shape(nl,na,2)
self.m = nn.ModuleList(nn.Conv2d(x, self.no * self.na, 1) for x in ch)
# output conv
self.inplace = inplace # use inplace ops (e.g. slice assignment)
def forward(self, x):
z = []  # inference output
for i in range(self.nl):
x[i] = self.m[i](x[i])  # conv
bs, _, ny, nx = x[i].shape  # x(bs,255,20,20) to x(bs,3,20,20,85)
x[i] = x[i].view(bs, self.na, self.no, ny, nx).permute(0, 1, 3, 4, 2).contiguous()

if not self.training:  # inference
if self.dynamic or self.grid[i].shape[2:4] != x[i].shape[2:4]:
self.grid[i], self.anchor_grid[i] = self._make_grid(nx, ny, i)

if isinstance(self, Segment):  # (boxes + masks)
xy, wh, conf, mask = x[i].split((2, 2, self.nc + 1, self.no - self.nc - 5), 4)
```

```
xy = (xy.sigmoid() * 2 + self.grid[i]) * self.stride[i] # xy
wh = (wh.sigmoid() * 2) ** 2 * self.anchor_grid[i] # wh
y = torch.cat((xy, wh, conf.sigmoid(), mask), 4)
else:  # Detect (boxes only)
xy, wh, conf = x[i].sigmoid().split((2, 2, self.nc + 1), 4)
xy = (xy * 2 + self.grid[i]) * self.stride[i] # xy
wh = (wh * 2) ** 2 * self.anchor_grid[i] # wh
y = torch.cat((xy, wh, conf), 4)
z.append(y.view(bs, self.na * nx * ny, self.no))

return x if self.training else (torch.cat(z, 1),) if self.export else
(torch.cat(z, 1), x)

def _make_grid(self, nx=20, ny=20, i=0,
torch_1_10=check_version(torch.__version__, '1.10.0')):
d = self.anchors[i].device
t = self.anchors[i].dtype
shape = 1, self.na, ny, nx, 2  # grid shape
y, x = torch.arange(ny, device=d, dtype=t),
torch.arange(nx, device=d, dtype=t)
yv, xv = torch.meshgrid(y, x, indexing='ij')
if torch_1_10 else torch.meshgrid(y, x) # torch>=0.7 compatibility
grid = torch.stack((xv, yv), 2).expand(shape) - 0.5
# add grid offset, i.e. y = 2.0 * x - 0.5
anchor_grid =
(self.anchors[i] * self.stride[i]).view((1, self.na, 1, 1, 2)).expand(shape)
return grid, anchor_grid
class Segment(Detect):
# YOLOv5 Segment head for segmentation models
def __init_(self, nc=80, anchors=(), nm=32, npr=256, ch=(), inplace=True):
super()._init_(nc, anchors, ch, inplace)
self.nm = nm # number of masks
self.npr = npr # number of protos
self.no = 5 + nc + self.nm  # number of outputs per anchor
self.m = nn.ModuleList(nn.Conv2d(x, self.no * self.na, 1) for x in ch)
 # output conv
self.proto = Proto(ch[0], self.npr, self.nm)  # protos
self.detect = Detect.forward
```

```
def forward(self, x):
p = self.proto(x[0])
x = self.detect(self, x)
return (x, p) if self.training else (x[0], p) if self.export else (x[0], p, x[1])


class BaseModel(nn.Module):
# YOLOv5 base model
def forward(self, x, profile=False, visualize=False):
return self._forward_once(x, profile, visualize)  # single-scale inference, train

def _forward_once(self, x, profile=False, visualize=False):
y, dt = [], [] # outputs
for m in self.model:
if m.f != -1: # if not from previous layer
x = y[m.f] if isinstance(m.f, int) else [x if j == -1 else y[j] for j in m.f]
# from earlier layers
if profile:
self._profile_one_layer(m, x, dt)
x = m(x) # run
y.append(x if m.i in self.save else None) # save output
if visualize:
feature_visualization(x, m.type, m.i, save_dir=visualize)
return x
def _profile_one_layer(self, m, x, dt):
c = m == self.model[-1] # is final layer, copy input as inplace fix
o = thop.profile(m, inputs=(x.copy() if c else x,),
verbose=False)[0] / 1E9 * 2 if thop else 0 # FLOPs
t = time_sync()
for _ in range(10):
m(x.copy() if c else x)
dt.append((time_sync() - t) * 100)
if m == self.model[0]:
LOGGER.info(f"{'time (ms)':>10s} {'GFLOPs':>10s} {'params':>10s}  module")
LOGGER.info(f'{dt[-1]:10.2f} {o:10.2f} {m.np:10.0f}  {m.type}')
if c:
LOGGER.info(f"{sum(dt):10.2f} {'-':>10s} {'-':>10s}  Total")

def fuse(self): # fuse model Conv2d() + BatchNorm2d() layers
LOGGER.info('Fusing layers... ')
for m in self.model.modules():
if isinstance(m, (Conv, DWConv)) and hasattr(m, 'bn'):
m.conv = fuse_conv_and_bn(m.conv, m.bn)  # update conv
delattr(m, 'bn') # remove batchnorm
m.forward = m.forward_fuse  # update forward
self.info()
return self

def info(self, verbose=False, img_size=640): # print model information
model_info(self, verbose, img_size)

def _apply(self, fn):
# Apply to(), cpu(), cuda(), half() to model tensors that are not
 parameters or registered buffers
self = super()._apply(fn)
m = self.model[-1]  # Detect()
if isinstance(m, (Detect, Segment)):
m.stride = fn(m.stride)
m.grid = list(map(fn, m.grid))
if isinstance(m.anchor_grid, list):
m.anchor_grid = list(map(fn, m.anchor_grid))
return self

class DetectionModel(BaseModel):
# YOLOv5 detection model
def __init_(self, cfg='yolov5s.yaml', ch=3, nc=None, anchors=None):
# model, input channels, number of classes
super()._init__()
if isinstance(cfg, dict):
self.yaml = cfg # model dict
```

```
else:  # is *.yaml
import yaml # for torch hub
self.yaml_file = Path(cfg).name
with open(cfg, encoding='ascii', errors='ignore') as f:
self.yaml = yaml.safe_load(f) # model dict

# Define model
ch = self.yaml['ch'] = self.yaml.get('ch', ch) # input channels
if nc and nc != self.yaml['nc']:
LOGGER.info(f"Overriding model.yaml nc={self.yaml['nc']} with nc={nc}")
self.yaml['nc'] = nc # override yaml value
if anchors:
LOGGER.info(f'Overriding model.yaml anchors with anchors={anchors}')
self.yaml['anchors'] = round(anchors) # override yaml value
self.model, self.save = parse_model(deepcopy(self.yaml), ch=[ch])
# model, savelist
self.names = [str(i) for i in range(self.yaml['nc'])]  # default names
self.inplace = self.yaml.get('inplace', True)

# Build strides, anchors
m = self.model[-1]  # Detect()
if isinstance(m, (Detect, Segment)):
s = 256 # 2x min stride
m.inplace = self.inplace
forward = lambda x: self.forward(x)[0]
if isinstance(m, Segment) else self.forward(x)
m.stride = torch.tensor([s / x.shape[-2] for x in forward(torch.zeros(1, ch, s, s))])
 # forward
check_anchor_order(m)
m.anchors /= m.stride.view(-1, 1, 1)
self.stride = m.stride
self._initialize_biases()  # only run once

# Init weights, biases
initialize_weights(self)
self.info()
LOGGER.info('')
```

```
def forward(self, x, augment=False, profile=False, visualize=False):
if augment:
return self._forward_augment(x)  # augmented inference, None
return self._forward_once(x, profile, visualize)  # single-scale inference, train

def _forward_augment(self, x):
img_size = x.shape[-2:] # height, width
s = [1, 0.83, 0.67] # scales
f = [None, 3, None] # flips (2-ud, 3-lr)
y = [] # outputs
for si, fi in zip(s, f):
xi = scale_img(x.flip(fi) if fi else x, si, gs=int(self.stride.max()))
yi = self._forward_once(xi)[0] # forward
# cv2.imwrite(f'img_{si}.jpg',
 255 * xi[0].cpu().numpy().transpose((1, 2, 0))[:, :, ::-1])
# save
yi = self._descale_pred(yi, fi, si, img_size)
y.append(yi)
y = self._clip_augmented(y)  # clip augmented tails
return torch.cat(y, 1), None  # augmented inference, train

def _descale_pred(self, p, flips, scale, img_size):
# de-scale predictions following augmented inference (inverse operation)
if self.inplace:
p[..., :4] /= scale  # de-scale
if flips == 2:
p[..., 1] = img_size[0] - p[..., 1]  # de-flip ud
elif flips == 3:
p[..., 0] = img_size[1] - p[..., 0]  # de-flip lr
else:
x, y, wh = p[..., 0:1] / scale, p[..., 1:2] / scale, p[..., 2:4] / scale  # de-scale
if flips == 2:
y = img_size[0] - y  # de-flip ud
elif flips == 3:
x = img_size[1] - x  # de-flip lr
```

```
p = torch.cat((x, y, wh, p[..., 4:]), -1)
return p
def _clip_augmented(self, y):
# Clip YOLOv5 augmented inference tails
nl = self.model[-1].nl # number of detection layers (P3-P5)
g = sum(4 ** x for x in range(nl)) # grid points
e = 1  # exclude layer count
i = (y[0].shape[1] // g) * sum(4 ** x for x in range(e))  # indices
y[0] = y[0][:, :-i] # large
i = (y[-1].shape[1] // g) * sum(4 ** (nl - 1 - x) for x in range(e)) # indices
y[-1] = y[-1][:, i:] # small
return y

def _initialize_biases(self, cf=None):
m = self.model[-1] # Detect() module
for mi, s in zip(m.m, m.stride):  # from
b = mi.bias.view(m.na, -1)  # conv.bias(255) to (3,85)
b.data[:, 4] += math.log(8 / (640 / s) ** 2) # obj (8 objects per 640 image)
b.data[:, 5:5 + m.nc] += math.log(0.6 / (m.nc - 0.99999))
if cf is None else torch.log(cf / cf.sum()) # cls
mi.bias = torch.nn.Parameter(b.view(-1), requires_grad=True)

Model = DetectionModel

class SegmentationModel(DetectionModel):
# YOLOv5 segmentation model
def __init_(self, cfg='yolov5s-seg.yaml', ch=3, nc=None, anchors=None):
super().__init__(cfg, ch, nc, anchors)
class ClassificationModel(BaseModel):
# YOLOv5 classification model
def __init_(self, cfg=None, model=None, nc=1000, cutoff=10):

# yaml, model, number of classes, cutoff index
super().__init__()
self._from_detection_model(model, nc, cutoff) if model is not None else
 self._from_yaml(cfg)
```

```
def _from_detection_model(self, model, nc=1000, cutoff=10)

# Create a YOLOv5 classification model from a YOLOv5 detection model
if isinstance(model, DetectMultiBackend):
model = model.model # unwrap DetectMultiBackend
model.model = model.model[:cutoff] # backbone
m = model.model[-1] # last layer
ch = m.conv.in_channels if hasattr(m, 'conv') else m.cv1.conv.in_channels

 # ch into module
c = Classify(ch, nc)  # Classify()
c.i, c.f, c.type = m.i, m.f, 'models.common.Classify' # index, from, type
model.model[-1] = c # replace
self.model = model.model
self.stride = model.stride
self.save = []
self.nc = nc

def _from_yaml(self, cfg):
# Create a YOLOv5 classification model from a *.yaml file
self.model = None



def parse_model(d, ch): # model_dict, input_channels(3)
# Parse a YOLOv5 model.yaml dictionary
LOGGER.info(f"\n{'':>3}{'from':>18}{'n':>3}{'params':>10}
{'module':<40}{'arguments':<30}")
anchors, nc, gd, gw, act = d['anchors'], d['nc'], d['depth_multiple'],
d['width_multiple'], d.get('activation')
if act:
Conv.default_act = eval(act) # redefine default activation, i.e.
Conv.default_act = nn.SiLU()
LOGGER.info(f"{colorstr('activation:')} {act}")  # print
na = (len(anchors[0]) // 2) if isinstance(anchors, list) else anchors
# number of anchors
no = na * (nc + 5) # number of outputs = anchors * (classes + 5)
```

```
layers, save, c2 = [], [], ch[-1]  # layers, savelist, ch out
for i, (f, n, m, args) in enumerate(d['backbone'] + d['head']):
# from, number, module, args
m = eval(m) if isinstance(m, str) else m  # eval strings
for j, a in enumerate(args):
with contextlib.suppress(NameError):
args[j] = eval(a) if isinstance(a, str) else a # eval strings

n = n_ = max(round(n * gd), 1) if n > 1 else n  # depth gain
if m in {
Conv, GhostConv, Bottleneck, GhostBottleneck, SPP, SPPF,
DWConv, MixConv2d, Focus, CrossConv,
BottleneckCSP, C3, C3TR, C3SPP, C3Ghost, nn.ConvTranspose2d,
DWConvTranspose2d, C3x}:
c1, c2 = ch[f], args[0]
if c2 != no: # if not output
c2 = make_divisible(c2 * gw, 8)

args = [c1, c2, *args[1:]]
if m in {BottleneckCSP, C3, C3TR, C3Ghost, C3x}:
args.insert(2, n) # number of repeats
n = 1
elif m is nn.BatchNorm2d:
args = [ch[f]]
elif m is Concat:
c2 = sum(ch[x] for x in f)
# TODO: channel, gw, gd
elif m in {Detect, Segment}:
args.append([ch[x] for x in f])
if isinstance(args[1], int): # number of anchors
args[1] = [list(range(args[1] * 2))] * len(f)
if m is Segment:
args[3] = make_divisible(args[3] * gw, 8)
elif m is Contract:
c2 = ch[f] * args[0] ** 2
elif m is Expand:
c2 = ch[f] // args[0] ** 2
else:
```

```
c2 = ch[f]
m_ = nn.Sequential(*(m(*args) for _ in range(n))) if n > 1 else m(*args) # module
t = str(m)[8:-2].replace('_main_.', '') # module type
np = sum(x.numel() for x in m_.parameters())  # number params
m_.i, m_.f, m_.type, m_.np = i, f, t, np  # attach index, 'from' index, type,
 number params
LOGGER.info(f'{i:>3}{str(f):>18}{n_:>3}{np:10.0f}  {t:<40}{str(args):<30}')
 # print
save.extend(x % i for x in ([f] if isinstance(f, int) else f) if x != -1)
# append to savelist
layers.append(m_)
if i == 0:
ch = []
ch.append(c2)
return nn.Sequential(*layers), sorted(save)

if___name___== '__main__':
parser = argparse.ArgumentParser()
parser.add_argument
('--cfg', type=str, default='yolov5s.yaml', help='model.yaml')
parser.add_argument
('--batch-size', type=int, default=1, help='total batch size for all GPUs')
parser.add_argument
('--device', default='', help='cuda device, i.e. 0 or 0,1,2,3 or cpu')
parser.add_argument
('--profile', action='store_true', help='profile model speed')
parser.add_argument
('--line-profile', action='store_true', help='profile model speed layer by layer')
parser.add_argument('--test', action='store_true', help='test all yolo*.yaml')
opt = parser.parse_args()
opt.cfg = check_yaml(opt.cfg)  # check YAML
print_args(vars(opt))
device = select_device(opt.device)

# Create model
im = torch.rand(opt.batch_size, 3, 640, 640).to(device)
model = Model(opt.cfg).to(device)
```

```
# Options

if opt.line_profile: # profile layer by layer
model(im, profile=True)
elif opt.profile: # profile forward-backward
results = profile(input=im, ops=[model], n=3)
elif opt.test:  # test all models
for cfg in Path(ROOT / 'models').rglob('yolo*.yaml'):
try:
_ = Model(cfg)
except Exception as e:
print(f'Error in {cfg}: {e}')

else:  # report fused model summary
model.fuse()
```

CHAPTER - 11
SCREENSHOTS

1. Training

The following images taken from the standard MOT16 dataset for the purpose of training the model. The model classifies various vehicle objects in the given images by numbering the objects. 0-Ambulance, 1–Bus, 2–Car, 3–Motorcycle, 4 –Truck.

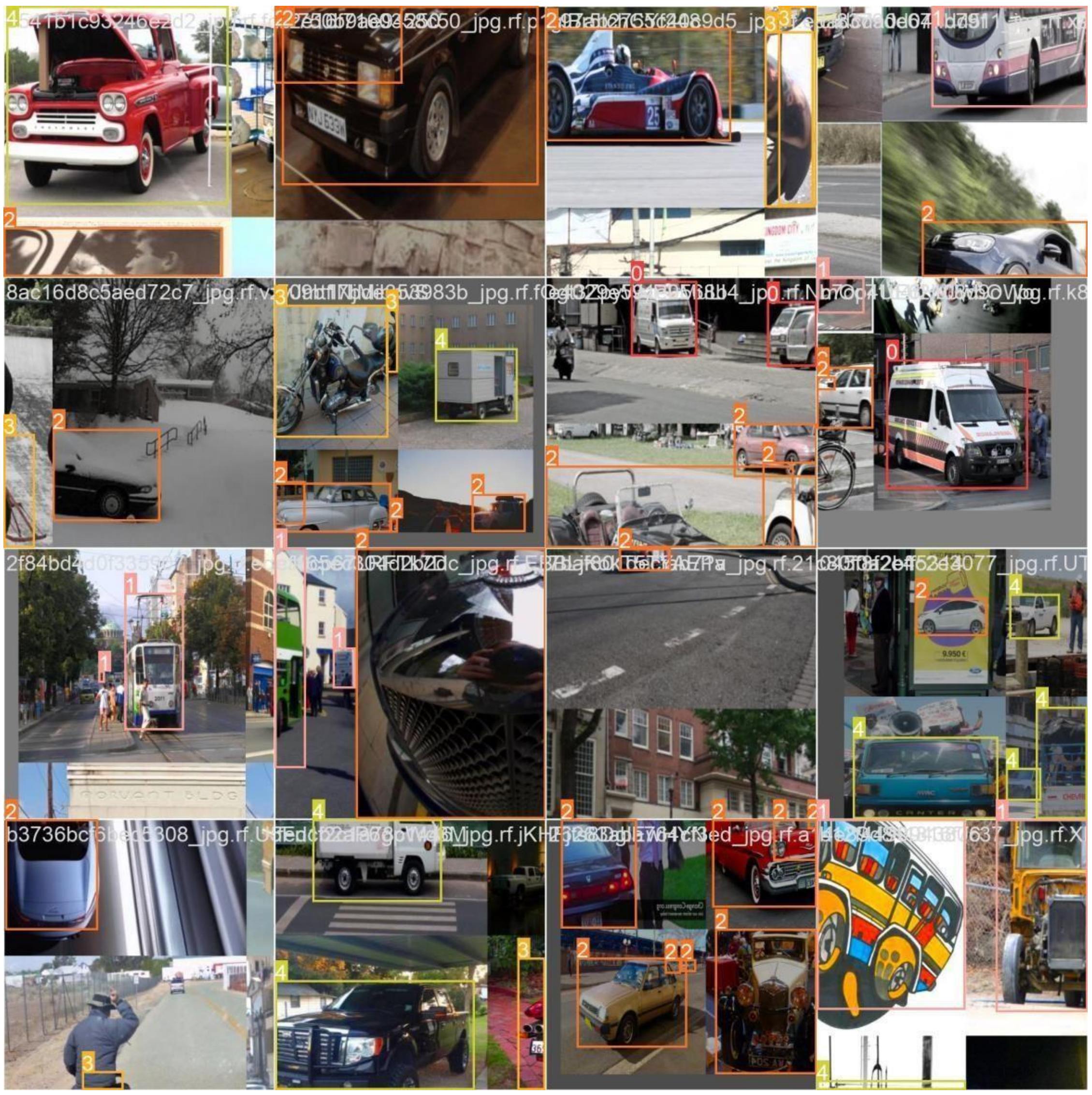


Fig 11.1 Training Screenshots

2. Validation Labels and prediction

The trained model is validated for the vehicle objects like Car, bus, ambulance, truck and motorcycle and labeling is done.

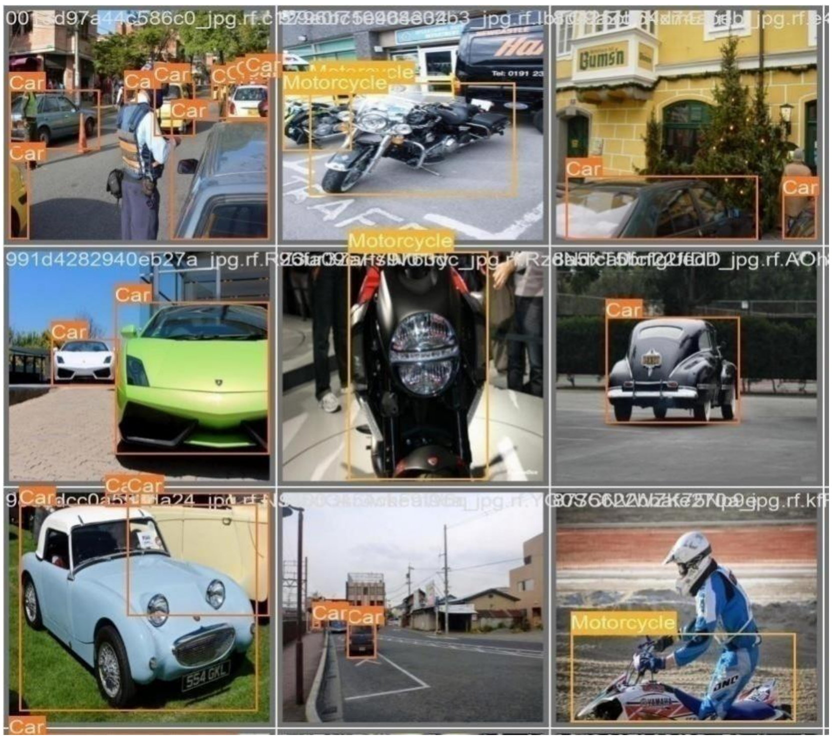


Fig 11.2 Validation labels

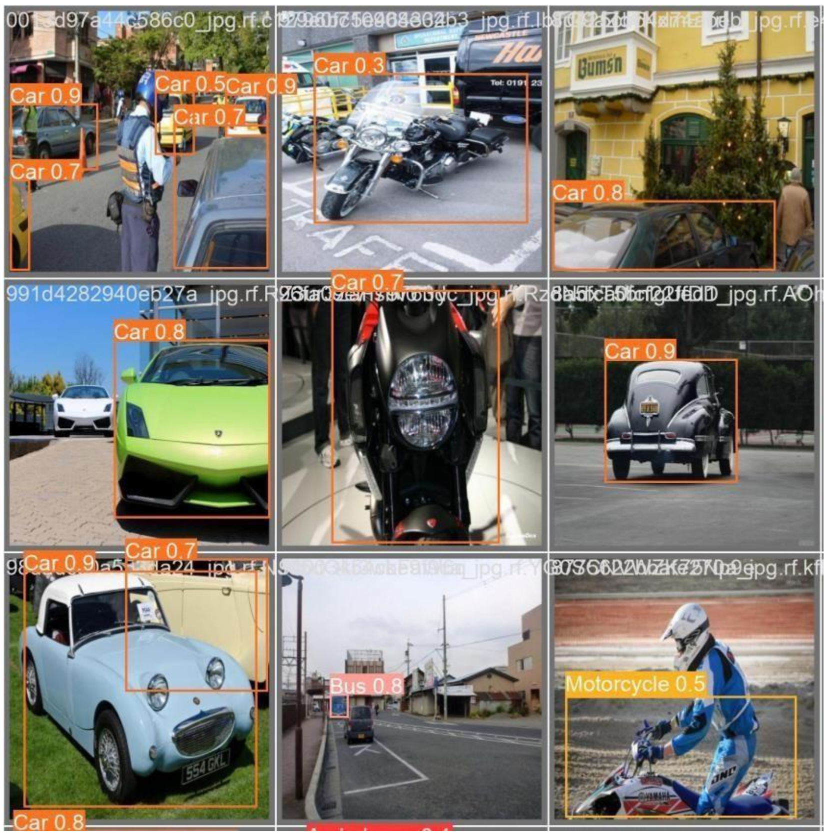


Fig 11.3 Validation Predictions

3. Testing for images

Now the trained system is used to detect the vehicles in testing dataset and respective confidence score is also given.

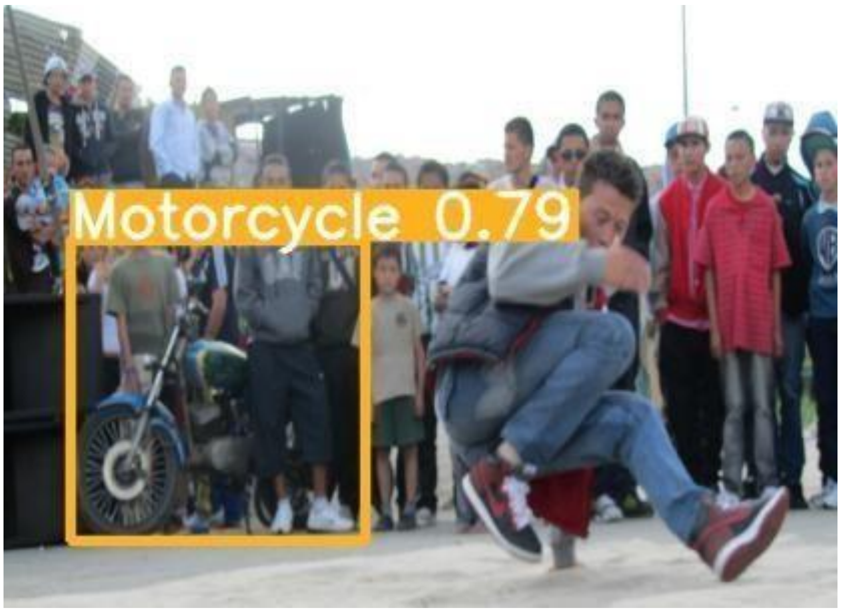


Fig 11.4 Testing for images

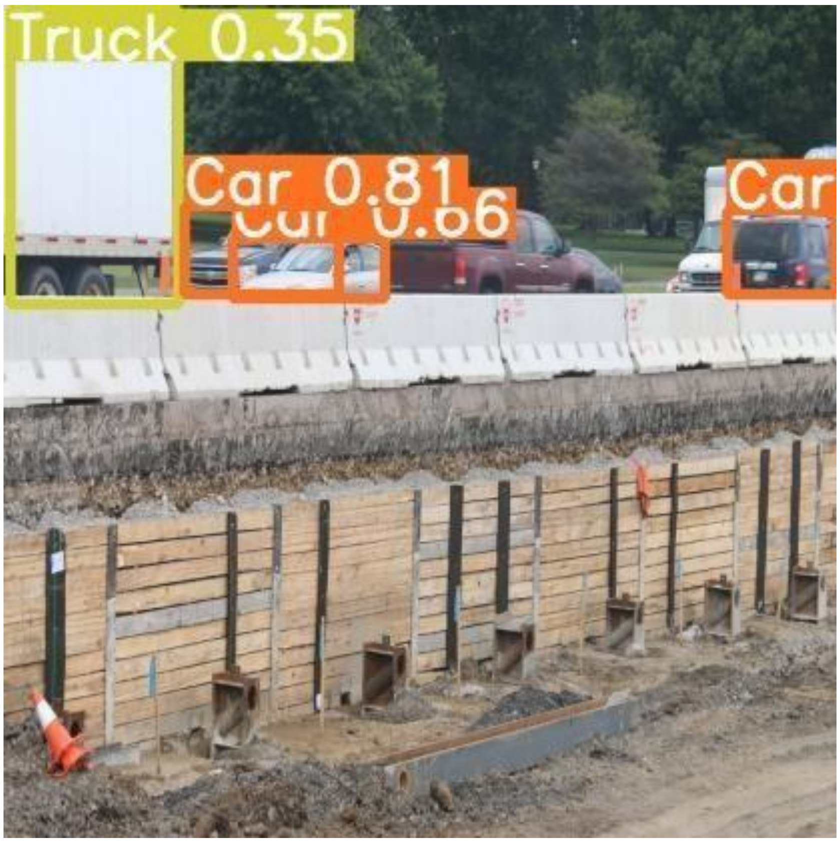


Fig 11.5 Testing for images

4. Testing for videos

The detection is applicable for videos as well. The trained model able to detect different classes of vehicles in videos and respective confidence score is shown

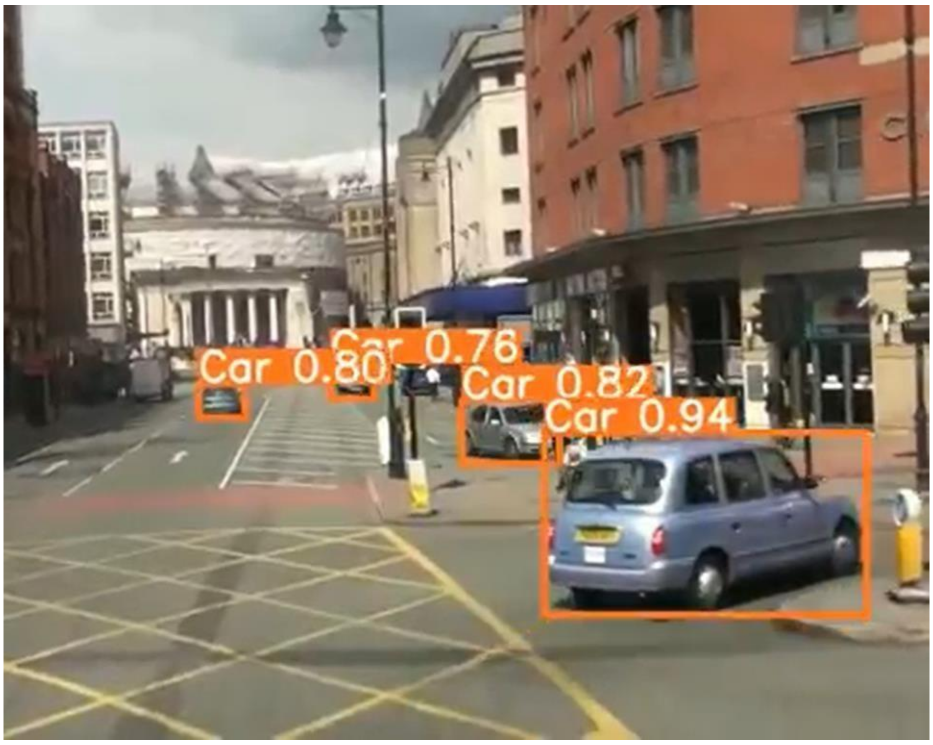


Fig 11.6 Testing for videos

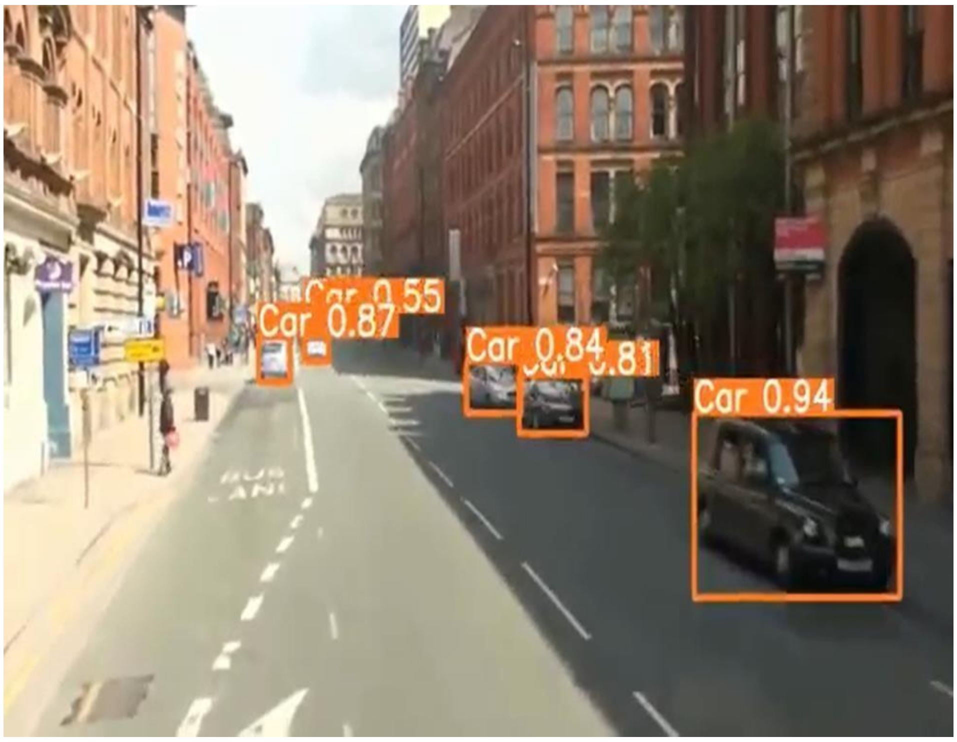


Fig 11.7 Testing for Videos

5. Training Graphs

This curve shows the confidence of the system in context of detecting that particular class of vehicle.

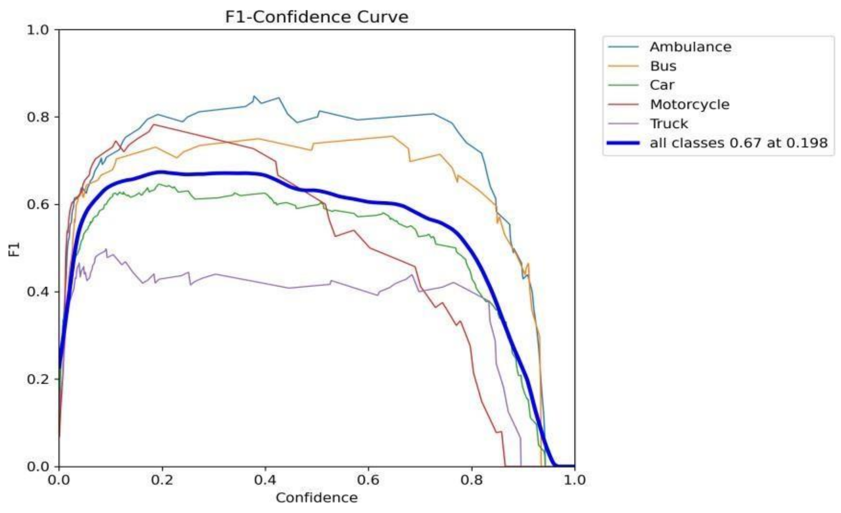


Fig 11.8 Confidence Graph

These set of graphs shows the minimization of losses and maximization of precision and recall while training.

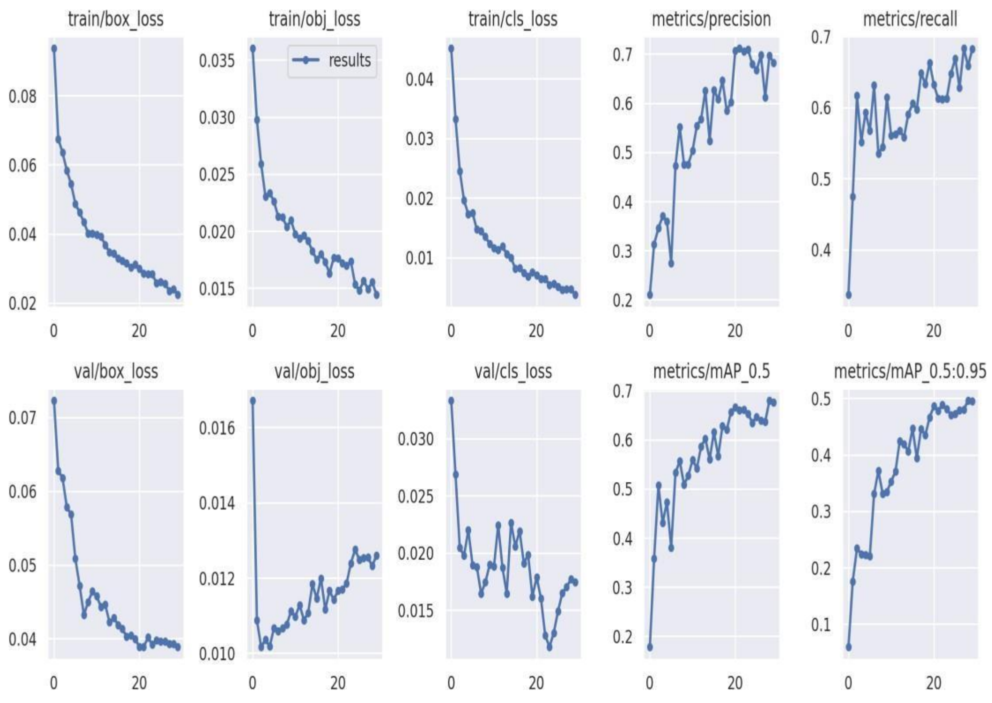


Fig 11.9 Precision Graphs

CHAPTER - 12
CONCLUSION AND FUTURE ENHANCEMENTS

This proposed system able to detect the different classes of vehicles accurately and create an impact in the different sectors like traffic management and surveillance etc. The future scope of this system is it can be used to train more than 5 classes of vehicles and make use of any advancement in YOLO algorithm.

The proposed system can be integrated with the cameras that are present at the traffic signals. Instead of giving a video as an input, this system can be enhanced to detect the vehicles directly by looking at the footage. Another scope is that, this system can be integrated with the routing systems like Google maps and Apple maps. This helps in re-routing some emergency vehicles to reach the destination on time by monitoring all routes and picking the route with less vehicles and avoiding routes with bigger trucks which can potentially cause congestion.

This system can also integrate with the smart autonomous vehicles to detect the vehicles around. These are some of the enhancements that can be potentially done to make use of this system to the fullest.

CHAPTER – 13

BIBLIOGRAPHY

1. Bochkovskiy, A.; Chien-Yao, W.; Liao, H.Y.M. YOLOv4: Optimal speed and accuracy of object detection. arXiv 2020, arXiv:2004.10934.
2. Zhao, H.; Li, Z.; Zhang, T. Attention Based Single Shot Multibox Detector. J. Electron. Inf. Technol. 2021, 43, 2096–2104.
3. Zheng, X.; Chen, F.; Lou, L.; Cheng, P.; Huang, Y. Real-Time Detection of Full-Scale Forest Fire Smoke Based on Deep Convolution Neural Network. Remote Sens. 2022, 14, 536.
4. de Haan, K.; Rivenson, Y.; Wu, Y.; Ozcan, A. Deep-Learning-Based Image Reconstruction and Enhancement in Optical Microscopy. Proc. IEEE 2020, 108, 30–50.
5. Zhan, W.; Sun, C.; Wang, M.; She, J.; Zhang, Y.; Zhang, Z.; Sun, Y. An improved Yolov5 real-time detection method for small objects captured by UAV. Soft Comput. 2022, 26, 361–373.
6. Xu, Z.; Huang, W.; Wang, Y. Multi-class vehicle detection in surveillance video based on deep learning. J. Comput. Appl. 2019, 39, 700–705.
7. S. Ren, K. He, R. Girshick, and J. Sun, “Faster r-cnn: towards real-time object detection with region proposal networks, ” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 39, pp. 1137–1149, 2017.
8. Yu Zhang; Zhongyin Guo ; Jianqing Wu ; Yuan Tian “Real-Time Vehicle Detection Based on Improved YOLO v5” September 2022, Sustainability 14(19): 12274
9. Fang, W., L. Wang, and P. Ren, Tinier-YOLO: A real-time object detection method for constrained environments. IEEE Access, 2019. 8: p. 1935-1944.
10. Miao, Y ,.et al. A Nighttime Vehicle Detection Method Based on YOLO v3. in 2020 Chinese Automation Congress (CAC). 2020. IEEE.

BASE PAPERS

2021 Third International Conference on Intelligent Communication Technologies and Virtual Mobile Networks (ICICV) | 978-1-6654-1960-4/20/$31.00 ©2021 IEEE | DOI: 10.1109/ICICV50876.2021.9388577

Object Detection for Autonomous Driving using YOLO [You Only Look Once] algorithm

Abhishek Sarda, Dr.Shubhra Dixit, Dr. Anupama Bhan

Dept. of ECE, ASET, Amity university,Noida, India Abhisheksarda0113@gmail.com , sdixit@amity.edu, abhan@amity.edu

***Abstract*—The field of autonomous driving is going to be the face of the automobile industry very soon. The number of accidents that take place because of human error currently is very high and it can be slashed to a huge extent with the advent of autonomous driving. One of the primary prerequisites and a huge part of autonomous driving is dependent on object detection through computer vision, this paper aims at aiding towards the field of autonomous driving by helping detect objects with the use of deep learning algorithms. Research work used state-of-the-art algorithm YOLO (you only look once) to detect different objects that appear on the road and classified into the category that they belong to with the help of bounding boxes. The weights of the YOLO v4 is utilized to custom train our model to detect the objects and the data will be collected from the open images dataset using its OIDv4 toolkit.**

Keywords—*YOLO, autonomous vehicles, object detection, computer vision.***

I. Introduction

When a child begins to walk the first thing that it tries to do is understand the surrounding around itself, similar is the case with autonomous vehicles, comprehension of the surroundings is of primary importance with regards to bringing the dream of bringing autonomous driving to reality. There are multiple sensors that aid for the purpose of understanding the surroundings but the most efficient ones are the camera sensors. On a comparative of all the sensors that exist, camera sensors tend to be the best performers at an efficient comprehension of the surrounding [1]. Comprehension of images containing multiple objects make it an extremely difficult task to incur especially when objects belonging to multiple classes are captured in the same image. Multiple companies have been successful in developing object detection algorithms such as NVIDIA [2], ViNotion, etc. which are being used by the autonomous driving giants like TESLA but information regarding these aren't widely available.

There are various research prospects pondering over different kinds of object detection algorithms. The primitive machine learning algorithms like SVMs, KNNs etc. have been taken over by the state of the art Deep CNNs. There are many algorithms like RCNN, Fast RCNN, etc. [3] which were initially used to quite an extent but they had to be replaced with the inception of the YOLO algorithm. The YOLO algorithm is much faster and equally good if not better with regards to the accuracy of detecting objects. It is trained on the MS COCO dataset and is capable of detecting objects in a general manner. In this paper we have trained the algorithm on a dataset of classes that we care about.

Autonomous cars have come a long way in the recent years, from cruze control to automated parking to auto pilot modes being there in a lot of cars currently. 90% of the accidents on the road happens due to human error and autonomous vehicles can help us cut through the human error and reduce risk of life significantly[4]. Consequently, there are a lot of areas to ponder upon before we can make that dream a reality. The main goal for an autonomous vehicle with regards to object detection is to correctly identify the objects that are present on the road like- car, truck, bus, traffic light, traffic sign, person, bicycles, building, etc. The primary requirements for object detection for autonomous driving vehicles are to make sure that they correctly identify the objects consistently with higher accuracies and to be able to identify them at a faster rate.

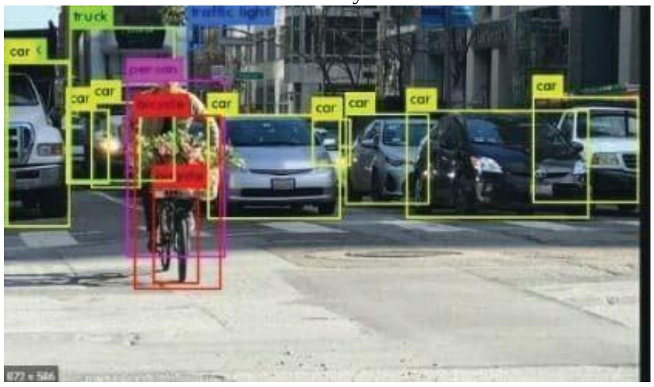


Fig 1: Objects being detected on a busy road

There are two pointers to be absolutely certain of when building an object detector: Firstly, to consistently maintain a high level of accuracy because if this aspect is not given its due importance the result could be extremely detrimental with regards to the end result [6], Secondly to be able to identify the objects on the images at a good pace. The major object detection algorithms pertaining to deep learning fall in one of two categories: Multi Stage Detector and Single Shot detector. The multi stage detectors identify the region proposals initially and then try to figure out the values of Bounding boxes. Single shot detectors do both the tasks simultaneously and hence get results at a faster rate. Both kinds of algorithms have certain drawbacks of their own and

that is what the researchers have been working on, in essence that the multi stage algorithms are undergoing experiments to make them give results at a faster rate while the single shot detector algorithms are working on getting them to produce higher levels of accuracy.
In this paper we have trained the YOLO algorithm on our custom classes to identify objects on the images with the use of bounding boxes.

II. Literature Review-

The models used for object detection in this day and age comprise of two portions, the primary portion which gets trained on the dataset of IMAGENET and is referred to as the backbone of the model and the second portion which is required for determining the coordinates of the bounding boxes and the class pertaining to the object in contention, and is also referred to as the head of the detection model. There are different backbones that are utilized depending on the platform that is being used i.e. GPU or CPU [7], if the detector is being run on the platform of the GPU the potential backbones in contention are RESNET, DENSENET,RESNEXT or the VGG NET. On the contrary, if the detector that is being run makes use of the CPU platform the backbones that then fall into contention are Mobile Net, Squeeze Net or Shuffle Net[8].
Talking about the head portion of the object detector it usually gets categorized into two different realms- multi stage detector and the single stage detector. The detectors commonly used in the former model are Fast RCNN, RCNN where as SSD along with YOLO are the most prominent of the latter models.
Anchor free detectors have also been procured in the recent time detectors of this kind include the CORNER net, Centre Net, etc. The detection models that are being developed in the current scenario tend to contain some layers between the head and their backbone, the layers combined together are termed as a NECK in the object detection model.
To quantify in simple words, object detection models are comprised of quite a parts which are articulated as follows:

a. Input- Image Pyramid, Images.
b. Neck- SPP, FPN, ASPP, etc.
c. Heads in one stage- YOLO, SSD
d. Heads in two stage- RCNN, Faster RCNN, etc.
e. Backbones- RESNEXT, VGG, etc.

The YOLO algorithm uses fully connected CNNs which are a distinctive kind of neural networks that generally deal with grid like structures [9]. The primary attribute of CNN that holds the maximum significance is the sharing of parameters. In a normal network distinctive weights correspond to their respective locations it is not quite the same in CNNs where members of the weights are utilized across the inputs which pertains to learning a set of weights for a particular location. This feature tends to have a significant role in making object detection feasible.

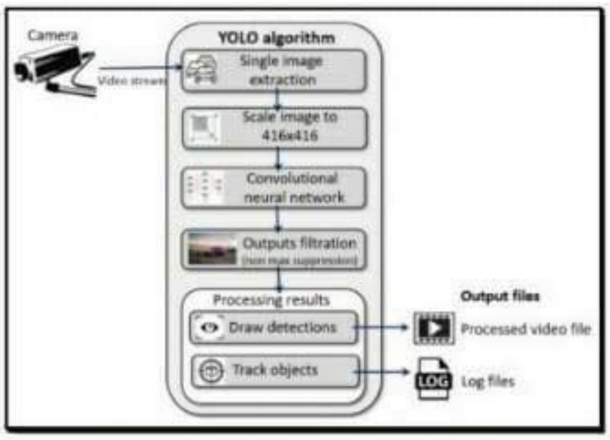


Fig 2: YOLO algorithm functioning

The manner in which a YOLO algorithm works is the image that is taken into consideration is divided into grid cells and the algorithm goes through the image only once and as a result of this the objects that are present in the given image are detected with the help of bounding boxes circumventing its perimeter. In the following image representation of the image in the model is shown. In our project, we had taken 8 different objects to train our model on, so it will be 13 dimensional one hot encoded vector. The first component represents the existence of the object, following four represent the bounding box and then the classes of objects.

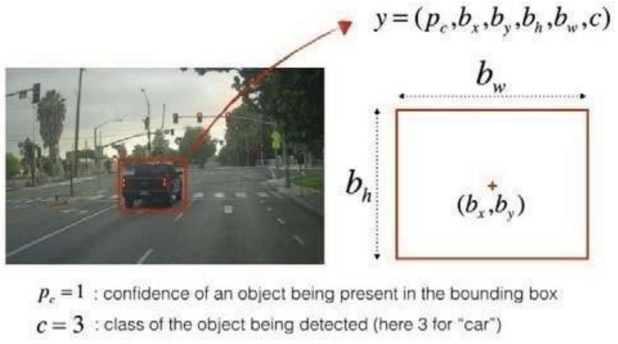

$y = (p_c, b_x, b_y, b_h, b_w, c)$
b_w
b_h
(b_x, b_y)
$p_c = 1$: confidence of an object being present in the bounding box
$c = 3$: class of the object being detected (here 3 for "car")

Fig 3: Representation of encoding

Anchor boxes are the boxes that are drawn for each grid cell to detect a certain number of objects, this project wave set the number of anchor boxes to 5 which means that the total number of anchor boxes in one image that are going to be drawn is going to be 5 x by the total number of grid cells [10]. This makes the number of anchor boxes to be huge and now we need to reduce the number of boxes that we can see in order to remove the clutter and being able to identify our object easily.
The next thing that we calculate is score of a class which is the multiplication of probability that a certain object exists x the class to which it belongs.
We set a certain threshold beyond which the value of the class score has to be in order to have a bounding box wrapped around it. In order to further reduce the number of boxes we apply something called as the non-max suppression which

basically removes all the boxes from the image which circumvent the objects from any portion apart from the mid-point of the object in consideration[12].

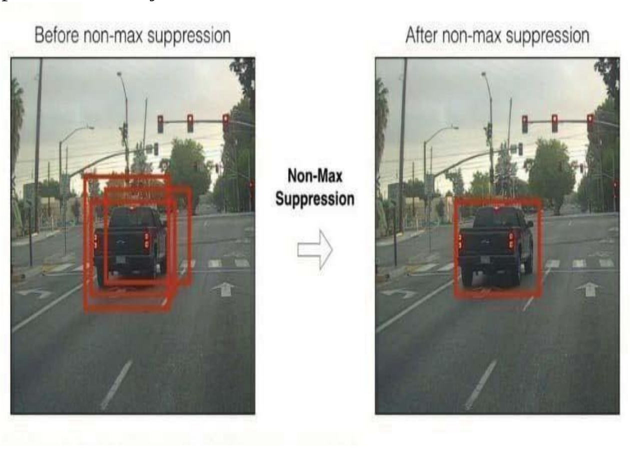


Fig 4: Non- max Suppression

III. YOLO architecture-

The backbone architecture that is used in YOLO v4 is CSPDarknet53[13]. It also consists of a NECK architecture of SPP along with PAN. The head architecture that is used is the YOLO v3[14].
Along with the architecture that is mentioned above the YOLO v4 makes use data augmentation architectures which are also termed as 'bag of freebies' because a change in the bof does create an extra cost on the training of the algorithm. The augmentation used for backbones are the drop block regularization, Mosaic data augmentation.
When the accuracy can be increased significantly by a very small increase in cost is termed as the 'bag of specials' [15]. The BOS used in backbone are Mish activation, CSP, Mi WPRC [16].
The BOF used for detector is- SAT, CmBN
The BOS used for detector is -Mish activation, SAM, PAN

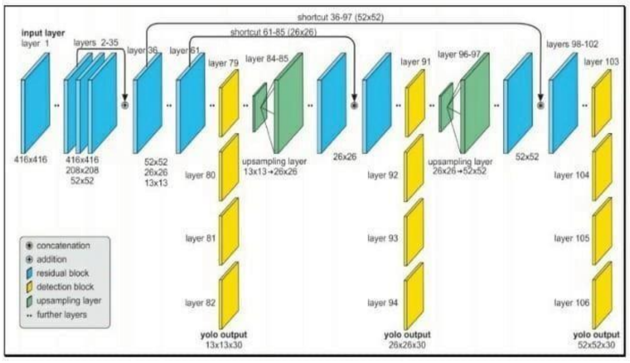


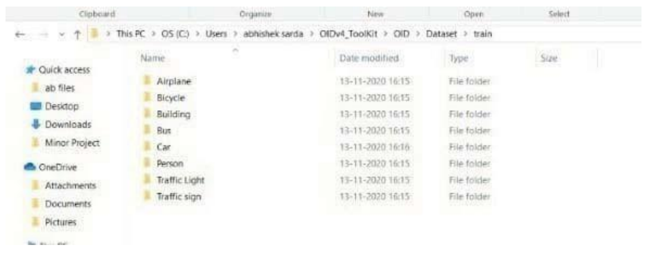
Fig 5: YOLO architecture

IV. Methodology adopted:

- Downloading the OID v4 tool Kit and setting it up for the purpose of preparing the dataset.
- Downloading the images that are required along with their labels
- Downloading the weights of the YOLO algorithm and get initial results on it.
- Setting up the directories and creating the YOLO annotations
- Uploading the dataset to the google drive and setting up the entire environment on colab.
- Preparing the code for custom training
- Implementing the YOLO algorithm by writing all the code and starting the training process.
- Getting the results on the custom dataset.

V. Data-

The data set that has been chosen has been taken from the open images data set which basically contains close to 6 million images.
From the open images data set images of certain categories have been extracted. Since the project that we are doing pertains to autonomous driving we want to choose classes of objects that are more likely to be found on the road, because of this reason we have chosen a total of 8 classes to custom train our YOLO algorithm. The classes are as follows- car, bicycle, person, traffic lights, traffic signals, bicycle, building, bus. The data that has been extracted has been extracted into two folds training and testing. The training set contains 1000 images for each class along with their labels while the testing set contains 400 images for each classes along with their labels. So our entire data set contains a total of 8000 images in the training data set and 3200 images in the testing data set which makes it a total of 11200 images.



This PC › OS (C:) › Users › abhishek sarda › OIDv4_ToolKit › OID › Dataset › train

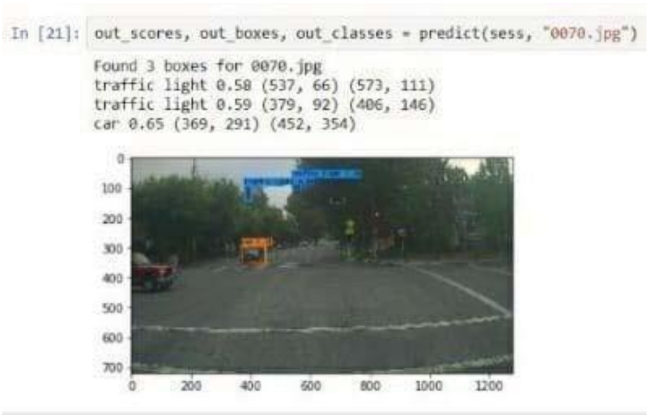
Name	Date modified	Type	Size
Airplane	13-11-2020 16:15	File folder	
Bicycle	13-11-2020 16:15	File folder	
Building	13-11-2020 16:15	File folder	
Bus	13-11-2020 16:15	File folder	
Car	13-11-2020 16:16	File folder	
Person	13-11-2020 16:15	File folder	
Traffic Light	13-11-2020 16:15	File folder	
Traffic sign	13-11-2020 16:15	File folder	

Fig 6: Dataset made from OID

VI. Initial Result-

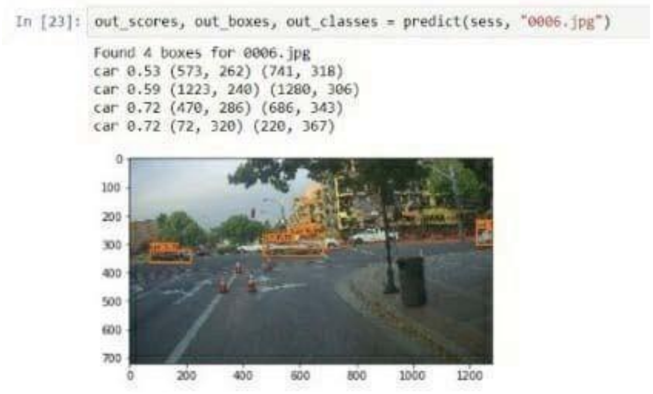
Firstly, we begin with cloning the darknet repository, and downloading the weights of the YOLO algorithm. Throughout the course of this project we have alluded to YOLO V4 as YOLO.
The YOLO algorithm has initially been trained on the MS Coco images data set and is trained to detect 80 classes of

objects. So before we begin with the custom training of our YOLO algorithm for objects on the road, we basically run the algorithm to test its accuracy on the basis of its pre trained weights.
Here is the initial result that we had received on testing the model based on its pre-trained weights on random images-



```
In [21]: out_scores, out_boxes, out_classes = predict(sess, "0070.jpg")
         Found 3 boxes for 0070.jpg
         traffic light 0.58 (537, 66) (573, 111)
         traffic light 0.59 (379, 92) (406, 146)
         car 0.65 (369, 291) (452, 354)
```

Fig 7: Result on pre-trained weights



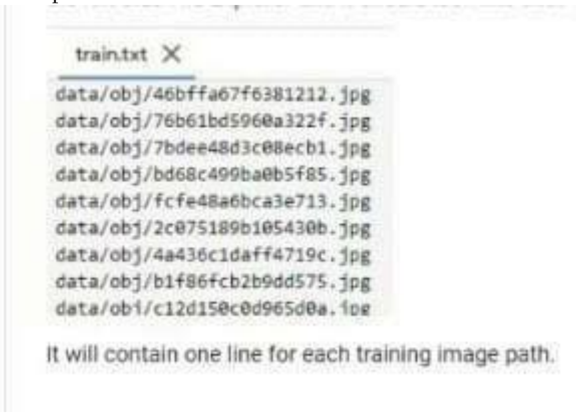
```
In [23]: out_scores, out_boxes, out_classes = predict(sess, "0006.jpg")
         Found 4 boxes for 0006.jpg
         car 0.53 (573, 262) (741, 318)
         car 0.59 (1223, 240) (1280, 306)
         car 0.72 (470, 286) (686, 343)
         car 0.72 (72, 320) (220, 367)
```

Fig 8: Result on prê-trained weights

VII. Custom Training-

Now we move on to begin the custom training for the classes that we have decided to train our custom YOLO algorithm on. Before we get ready with the data set there are some initial pre-processing that needs to be done.
The YOLO algorithm detects objects with the help of bounding boxes around the objects that it detects so we want to convert the labels of images in the training and testing data set into the YOLO format where it has details of bounding boxes which are known and annotations of the images. After the annotations have been generated and converted into the YOLO format, we incorporate all the data
which includes the annotations and the images for training and testing into two separate folders respectively. Now we zip the two folders and upload it to our Google drive so that we can use the custom data set that we have created. The next step is to copy the data set on our cloud virtual machine and unzip the data set which contains both the training and the testing data.

Now we move towards creating the config file for the purpose of training our data set. After creating this file we upload this on our cloud VM as well.
The next step is to create a file that contains names of all the objects that we are going to train our YOLO algorithm on and another file that contains data on all those objects.
After uploading both the files on a cloud virtual machine we move on to generate text for our training and testing data set. After the text has been generated it is converted into a python file and uploaded to the virtual machine.



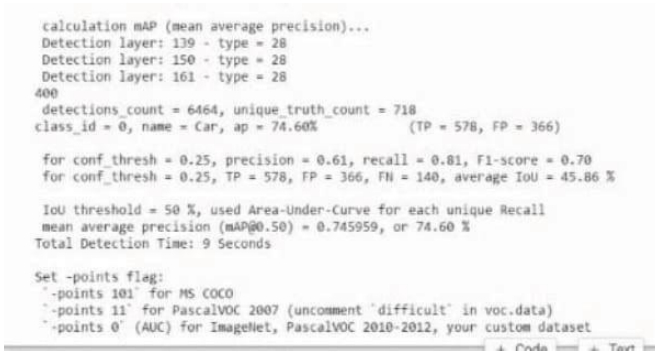
train.txt ✕

```
data/obj/46bffa67f6381212.jpg
data/obj/76b61bd5960a322f.jpg
data/obj/7bdee48d3c08ecb1.jpg
data/obj/bd68c499ba0b5f85.jpg
data/obj/fcfe48a6bca3e713.jpg
data/obj/2c075189b105430b.jpg
data/obj/4a436c1daff4719c.jpg
data/obj/b1f86fcb2b9dd575.jpg
data/obj/c12d150c0d965d0a.jpg
```

It will contain one line for each training image path.

Fig 9: Text generated

At this point all the data that was required has been created and our algorithm is finally ready for the purpose of custom training.
Before we begin training we create a folder in a drive with the name called backup that is basically linked to our training kernel and helps us save weights after every thousand epochs so that our trained weights are not lost if training gets stopped due to some reason.
Here is the accuracy that we have received after training our YOLO algorithm for closed to 12 hours using all the data that we had-



```
 calculation mAP (mean average precision)...
 Detection layer: 139 - type = 28
 Detection layer: 150 - type = 28
 Detection layer: 161 - type = 28
400
 detections_count = 6464, unique_truth_count = 718
class_id = 0, name = Car, ap = 74.60%           (TP = 578, FP = 366)

 for conf_thresh = 0.25, precision = 0.61, recall = 0.81, F1-score = 0.70
 for conf_thresh = 0.25, TP = 578, FP = 366, FN = 140, average IoU = 45.86 %

 IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
 mean average precision (mAP@0.50) = 0.745959, or 74.60 %
Total Detection Time: 9 Seconds

Set -points flag:
 `-points 101` for MS COCO
 `-points 11` for PascalVOC 2007 (uncomment `difficult` in voc.data)
 `-points 0` (AUC) for ImageNet, PascalVOC 2010-2012, your custom dataset
```

Fig 10: Result obtained on colab notebook

VIII. Result-

Objects	Car, Person, building, Bicycle, Traffic Light, Traffic Sign, Building, Bus
mAP	74.6%
Input size	416*416

F1-score	0.70
Recall	0.81

Table 1: Final Result

Epoch	Precision	Recall	F1 Score	mAP
500	0.3	0.4	0.35	37.3%
600	0.4	0.5	0.48	48.76%
1000	0.61	.81	0.7	74.59%
2000	0.64	.81	0.71	77.8%
3000	0.66	.81	0.73	79.58%

Table 2: Epoch wise results

After training our YOLO algorithm on our custom dataset we have received the mAP value of 74.6% which is pretty impressive as the training has been done on 8 different classes on a total of 1000 training images for all the 8 classes and this is the result that we have taken to be our final result as the improvement in the model started to decrease and had almost become constant. As represented in the Table 2 , we can barely witness any increase in precision with increase in epochs. So in order to prevent over-fitting we consider the weights up to 1000 epochs only. There is only 5% increase in the mAP value from the 1000 epoch mark to the 3000 epoch mark, which means the model stops improving after 1000 iteration and starts to overfit on th training set to get a better accuracy.

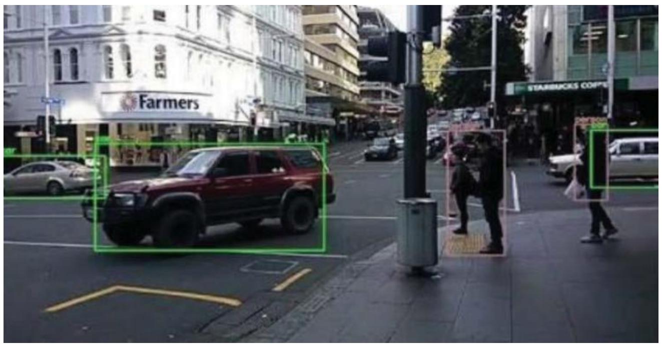


Fig 11: Result on image after custom training

IX. Conclusion -

For the purpose of object detection, it is imperative that we have a very high accuracy rate along with a faster speed and there cannot be any loop hole in this. In this paper we have successfully trained our YOLO model to aid in the field of autonomous driving through the camera sensors for the purpose of object detection, our trained model is capable of detecting various objects that appear on the road by creating a bounding box around the object and captioning it with the class category that it belongs to. In our model, we still have incurred some false positives and false negatives which we have got to reduce because it can we very detrimental for application if adversaries like this exist. False positives can also cause fatal accidents and that is also something we have tried to curb down by using the sufficient images and proper training.

X. References:

[1] Felzenszwalb, P. F., Girshick, R. B., McAllester, D., & Ramanan, D. (2010). Object detection with discriminatively trained part -based models. IEEE transactions on pattern analysis and machine intelligence, 32(9), 1627-1645.

[2] Goodfellow, I., Bengio, Y., Courville, A., & Bengio, Y. (2016). Deep learning (Vol. 1). Cambridge: MIT press.

[3] Girshick, R. (2015). Fast r-cnn. In Proceedings of the IEEE international conference on computer vision (pp. 1440-1448).

[4] Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. In Advances in neural information processing systems (pp. 91 -99).

[5] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 779 -788).

[6] Redmon, J., & Farhadi, A. (2018). Yolov3: An incremental improvement. arXiv preprint arXiv:1804.02767. [7] Springenberg, J. T., Dosovitskiy, A., Brox, T., & Riedmiller, M. (2014). Striving for simplicity: The all convolutional net. arXiv preprint arXiv:1412.6806.

[8] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

[9] Lin, T. Y., Dollár, P., Girshick, R. B., He, K., Hariharan, B., & Belongie, S. J. (2017, July). Feature Pyramid Networks for Object Detection. In CVPR (Vol. 1, No. 2, p. 4).

[10] Hartigan, J. A., & Wong, M. A. (1979). Algorithm AS 136: A kmeans clustering algorithm. Journal of the Royal Statistical Society. Series C (Applied Statistics), 28(1), 100-108.

[9] Navaneeth Bodla, Bharat Singh, Rama Chellappa, and Larry S Davis. Soft-NMS–improving object detection with one line of code. In Proceedings of the IEEE International Conference on Computer Vision (ICCV), pages 5561–5569, 2017. 4

[10] Zhaowei Cai and Nuno Vasconcelos. Cascade R-CNN: Delving into high quality object detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 6154–6162, 2018. 12

[11] Jiale Cao, Yanwei Pang, Jungong Han, and Xuelong Li. Hierarchical shot detector. In Proceedings of the IEEE International Conference on Computer Vision (ICCV), pages 9705–9714, 2019. 12

[12] Ping Chao, Chao-Yang Kao, Yu-Shan Ruan, Chien-Hsiang Huang, and Youn-Long Lin. HarDNet: A low memory traffic network. Proceedings of t he IEEE International Conference on Computer Vision (ICCV), 2019. 13

[13] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. DeepLab: Semantic image segmentation with deep convolutionalnets, atrous convolution, and fully connected CRFs. IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI), 40(4):834–848, 2017. 2, 4

[14] Pengguang Chen. GridMask data augmentation. arXiv preprint arXiv:2001.04086, 2020. 3

[15] Yukang Chen, Tong Yang, Xiangyu Zhang, Gaofeng Meng, Xinyu Xiao, and Jian Sun. DetNAS: Backbone search for object detection. In Advances in Neural Information Processing Systems (NeurIPS), pages 6638–6648, 2019. 2

[16] Jiwoong Choi, Dayoung Chun, Hyun Kim, and Hyuk -Jae Lee. Gaussian YOLOv3: An accurate and fast object detector using localization uncertainty for autonomous driving. In Proceedings of the IEEE International Conference on Computer Vision (ICCV), pages 502–511, 2019. 7

[17] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. R-FCN: Object detection via region-based fully convolutional networks. In Advances in Neural Information Processing Systems (NIPS), pages 379–387, 2016. 2

[18] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 248–255, 2009. 5

2019 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, SustainableComputing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)

An Improved Vehicle Detection Algorithm based on YOLOV3

1st Xiaoqing Sun
School of Computer and Information
Hohai University
Nanjing, China
alukaccap@163.com

2nd Qian Huang*
School of Computer and Information
Hohai University
Nanjing, China
huangqian@hhu.edu.cn

3rd Yanping Li
School of Computer and Information
Hohai University
Nanjing, China
liyp8023yz@163.com

4th Yuan Huang
School of Computer and Information
Hohai University
Nanjing, China
huan99uan@163.com

***Abstract*—With the development of science and technology, the application of vehicle object detection in intelligent video monitoring and vehicle assisted driving has become more extensive. Traditional vehicle object detection algorithms have some inefficiencies in generalization ability and recognition rate. To solve this problem, this paper proposes an improved vehicle detection method based on YOLO(You Only Look Once)V3. Our model improves the YOLOV3 algorithm, integrating the label smoothing, and adopts K-means++ algorithm to analyze the data aggregation and avoid the disappearance of the gradient, make our train model has better generalization ability. Finally, we use the UA-DETRAC data set for training, and the experimental results showed that our method has a better detection speed and a higher recognition rate for vehicle detection, which is 6% mAP higher than the original YOLOV3 algorithm.**

***Index Terms*—object detection, label smoothing, K-means++**

I. Introduction

With the rapid development of economy and the gradual improvement of people's quality of life, more and more families own private cars, which leads to the rapid growth of urban traffic vehicles, and traffic management is faced with great challenges. Therefore, rapid detection and identification of vehicles in traffic images have become an important task in urban traffic management and a research focus in the field of computer vision.

This work is supported in part by the National Key Research and Development Program of China under grant no. 2018YFC1508603 and no. 2018YFC0407905, the Natural Science Foundation of Jiangsu Province under grant no. BK20170893, the key research and development program of Jiangsu Province under grant no. BE2016904, the Demonstration Course Project of Ideology and Politics in Hohai University under grant no. 2018B06, and the National Natural Science Foundation of China under grant no. 61300122.

When people glance at an image, they can find what the objects are, and where they are. Such as human visual system mentioned abovečňvehicle object detection is to identify vehicle objects in different scenes. Since the 1970s, researchers have been conducting theoretical research. With the deepening of the researchčňvehicle object detection have been used for commercial operation in 1990s. Although vehicle object detection technology has been developed for nearly 50 years, there are still some difficult problems to solve. For example, the interference of complex background exists in the actual application scene, and the overlap of vehicles is caused by a large number of vehicles appearing in the picture at the same time when the traffic is heavy. In particular, the impact of technical rain, snow, night and other weather conditions will reduce the accuracy of vehicle object detection, which makes vehicle object detection become a difficulty in the field of computer vision.

Vehicle detection belongs to the scope of object detection. Before deep learning is widely applied in the field of computer vision, researchers conduct object detection by establishing some mathematical models based on certain empirical knowledge. The traditional object detection method is generally divided into three stages: firstly, select candidate areas on a given image by using sliding windows of different sizes, then extract the characteristics of these candidate regions, finally, the trained classifier is used for classification. By extracting manual features, such as HOG [1], SIFT [2], classifiers such as Support Vector Machine(SVM) [3] and Adaboost [4] were mainly used in the classification stage. However, there are two main problems in traditional object detection: 1) region selection based on sliding window is based on exhaustive strategy, with high time complexity and window redundancy; 2) the featur manually extracted by researchers based on experience are not robust to diversity changes, and the generalization ability is poor.

At the same time, lighting, angle of view and some changes inside the vehicle all will affect the video and the vehicle objects in the picture, making the detection results unsatisfactory. These disadvantages limit the application of detection algorithm. With the excellent performance of deep learning in the field of image processing, deep neural network begins to subvert the traditional feature extraction method. Through a large amount of data training, the model can autonomously learn useful features [5]. Currently, the mainstream object detection methods based on deep neural network are mainly divided into two stages, the two-stage method and the one-stage method, namely the method based on regional suggestions and the method based on regression: 1) two-stage methods, such as R-CNN [6], SPP-NET [7], Fast R-CNN [8], R-FCN [9]; 2) tem one-stage methods, such as YOLO [10], YOLOv2 [11], SSD [12], and YOLOV3 [13].

Different from the traditional machine learning method, the deep learning methods have good adaptability to the influence of image geometric transformation, deformation and illumination, overcomes various difficulties caused by vehicle problems, and has certain generalization ability.

The proposed object detection algorithm R-CNN introduced CNN into object detection for the first time, which greatly improved the object detection effect and became the mainstream in the object detection field at that time. R-CNN uses convolution neural network(CNN) to extract the image features, selective search method is used to extract [14] suggestion box, this can have much improved compare to traditional machine learning methods, but the R-CNN input image size is fixed, training steps trivial, test speed slower and cannot be updated in real time. SPP-NET proposed pyramid pooling layer to solve the problem of fixed input layer size of deep network. However, in the fine-tuning stage, SPP-NET only updated the full connection layer behind pyramid pooling layer and could not reverse propagation error. Moreover, the characteristics of training needed to be stored in disk, which limited the improvement of detection accuracy and speed. Fast R-CNN was proposed to solve the problem of slow test and training speed of R-CNN and SPP-NET, and multi-task learning was introduced to integrate multiple steps into one model. Meanwhile, multi-task loss function was introduced to fast-rcnn, which made the training and testing of the whole network very convenient and greatly improved the calculation speed. Based on Fast-RCNN, RPN(Region Proposal Networks) were added to the backbone network to achieve higher accuracy in VOC data set. R-CNN series have high detection accuracy, but its detection speed is slow, which is difficult to meet the requirement of real-time object detection.

Because YOLO series models are widely used in industry, this paper chooses YOLOV3 model for vehicle object detection.Although YOLOv3 achieves a good real-time detection effect, it is still not enough for vehicle object detection. Therefore, this paper proposes a series of algorithm improvements based on YOLOV3 model. In order to constrain the model and reduce the degree of over-fitting, the tag smoothing method [15] is introduced to improve the performance of the model. K-means++ is using for clustering analysis to determine the optimal clustering number and the corresponding width and height, and then the anchor parameters in YOLOV3 algorithm were modified accordingly. To achieve high precision and fast vehicle target detection framework.Finally, the UA-DETRAC vehicle data set was used to verify and analyze our model.

The remaining sections of this paper are organized as follows: Section 2 describes the basic object detection models including a brief introduction about the YOLOV3 algorithm, the Darknet framework. Section 3 introduces the tag smoothing method and YOLOV3 network which adopts a multi-scale prediction framework. Section 4 presents experimental verification to validate the effectiveness of the proposed detection model. Finally, conclusions are summarized in Section 5.

II. RELATED WORK

A. YOLOV3 Detection Process

Step1: First of all, divided the whole image into $S * S$ grid, each cell contains a prediction B frame (bounding boxes), and each prediction frame contains five parameters, namely(x,y,w,h,$object\_conf$). The(x,y) to predict border center and the corresponding to the offset value of grid boundary. The width (w,h) are the ratio of the width and height of the frame to the width and height of the whole image, objectconf for object confidence score, the frame includes a confidence value of the object, As shown in equation (1).

$$object\_conf = Pr(object) * IOU^{truth}_{pred} \tag{1}$$

The p_r(object) represents the cell corresponding to the prediction border contains the object, with a value of 1 for inclusion and 0 for non-inclusion. $IOU^{truthshows}_{pred}$ represents the intersection over union of the predicted border and the real border.

$$\begin{aligned} object\_conf &= Pr(class|object) * Pr(class) * IOU^{truth}_{pred} \\ &= Pr(class) * IOU^{truth}_{pred} \end{aligned} \tag{2}$$

Step2: Convolutional neural network is used to extract features and make predictions. When the number of categories in the network is C and the conditional existence probability of the objects in the grid is ($P_r(class|object)$), the probability of obtaining a certain category is $P_r(class_i)$. And the product of the probability of a certain class and corresponding intersection ratio is the confidence value of the class, as shown in equation(2).

Step3: Filter the predicted border by non-maximum suppression (NMS), remove the duplicate border, and output the final predicted result. The design of loss function uses the mean square value and error value of the dimension vector $S * S * (B\ 5+C)$ and the truth value of the image. Finally, it achieves the purpose of optimizing the model structure. However, there is no object in many grids, the loss function sets different proportional coefficients for the prediction box and without the object respectively to balance the difference [21]. In addition, the loss coefficients of the bounding box and the loss coefficients of the judgment category are also added, which can give a higher weight to the loss coefficients of the objects included in the bounding box.

B. Darknet-53 Network Structure

An excellent feature extraction network is the core of object detection algorithm. As a backbone network, its performance and precision directly affect the performance of YOLOv3 algorithm. Darknet-53 is a feature extraction network proposed by YOLOv3 creator and applied in practice, compared with other feature extraction networks in the same period, Darknet-53 still has excellent performance.

In deep learning, the deeper the level of the network is, the gradient of back propagation in the network will become unstable along with the multiplication and become extremely large or small. It is easy for the gradient to disappear or explode, leading to the degradation of the network.Even using the Batch Normalization methods or using Xaiver initialization, the effect is still not satisfactory. In 2015, Kaiming He proposed ResNet in the ILSVRC competition [16]. Darknet-53 from layer 0 to layer 74, there are 53 convolution layers in total, and the rest are Res layers.Darknet-53 by introducing res layer, the whole network is divided into several small ResNet structure unit, through step by step to learn residual to control the spread of the gradient. ResNet principle as shown in fig.1, its main idea is increased in the network structure direct channel, passing the original output of one layer directly to a later layer, through jump connection, making a residual block information flows to the next residual unimpeded, improving the circulation of information, and also avoid the too deep with network caused the disappearance of the gradient and degradation problems [17].

In ResNet, if x_l and x_l+1 are used to represent the input and output of the l layer respectively, W_l represents the weight of the l layer, and F represents the residual function of the layer, then the relationship between x_l and $x_l + 1$ can be expressed by equation(3):

$$x_{l+1} = x + F(x_l, W_l) \tag{3}$$

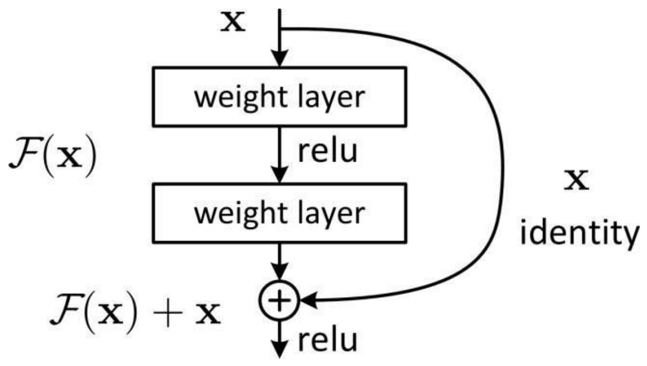

x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Fig. 1. Residual block diagram.

If the network learns the L layer in such a structure, x_L represents the input of the L layer, and the relationship between x_L and x_l can be expressed as equation(4):

$$x_L = x_l + \sum_{i=l}^{L-1} F(x_i, W_l) \tag{4}$$

Thus, the gradient of loss function in the reverse transfer process can be obtained as shown in equation(5):

According to the two terms in brackets in formula(5), the number 1 ensures that the gradient can be transmitted loss less. Although the size of the second term is determined by the weight of the network, no matter how small the item is, the gradient will not disappear. It can be seen that ResNet's learning of the original input information is easier and more accurate.

$$\frac{\partial Loss}{\partial x_l} = \frac{\partial Loss}{\partial x_L} * \frac{\partial x_L}{\partial x_l} = \frac{\partial Loss}{\partial x_L} * \left(1 + \frac{\partial}{\partial x_L} \sum_{i=1}^{L-1} F(x_i, W_i)\right) \tag{5}$$

III. PROPOSED SCHEME

The YOLO model has been updated for three times, and the network structure of YOLOV3 is quite mature, which has become the most widely used object detection algorithm in the industry. Therefore, our work mainly focus on the application of YOLOV3 model optimization to vehicle target detection, and creatively introduce tag smoothing and K-means++ to improve the detection accuracy, so that YOLOV3 model can achieve better results in vehicle object detection.

A. Label Smoothing

At the beginning of the design of YOLOV3 model, in order to pursue the real-time detection, multi-label classification is adopting for category prediction, which reduced the accuracy to some extent.On this basis, we introduce LSR(Label Smoothing Regularization) constraint model to classify targets, reduce the degree of overfitting of the model, and improve the accuracy of detection.

For each object, the detection network usually uses the softmax function to calculate the probability distribution of all classes:

$$P_i = \frac{\exp^{Z_i}}{\sum_i \exp^{Z_i}} \tag{6}$$

Where Z_i is a non-standardized logics directly from the last linear layer, used for classification prediction. For the target detection during training, we only modified the classification loss by comparing the output distribution p with the real value distribution q and the cross entropy:

$$L = \sum_i q_i * \log p_i \tag{7}$$

The q is usually a mono-thermal distribution, where the correct class has probability 1, and all other classes have 0. However, the softmax function can only approach this distribution at $Z_i >> Z_j$, but $\forall j \quad i$ never gets there. This will cause the model to believe the category of prediction too much, unable to guarantee the generalization ability of the model, and easy to cause over-fitting.

LSR can prevent the model from over-concentrating the predicted value on the larger probability category and assigning some probability to other smaller probability categories, so as to realize the constraint on the model and reduce the degree of overfitting of the model.

$$q_i' = (1 - \epsilon) * q_i + \frac{\epsilon}{K} \tag{8}$$

Where K is the total number of classes and ϵ is a small constant.The technique reduces the confidence of the model through maximum and minimum logits (unnormalized generalization Rate) to measure the difference between.

B. Optimization of Anchor Box Dimension Clustering

The k-means method is to manually specify the preliminary clustering center, which is more suitable for detecting images with small changes in the target position. In the road monitoring pictures, the position of vehicles is relatively random, and the detection accuracy will be largely lost if the initial clustering center is directly set. Therefore, K-means++ algorithm is adopted to optimize the selection of anchor frame and improve the accuracy of vehicle object detection.

In YOLOV2, the concepts of anchor points and preselected regions are added by referring to the ideas of Faster-RCNN. In YOLOV3, the author reserved anchor points and candidate regions. The location, number and size of suitable candidate areas should be selected to achieve the highest accuracy in the shortest possible time. In YOLOV3, K-means algorithm is used to cluster through ground true of the data set to find the rule of occurrence width and height of anchor box. Meanwhile, K is specified in advance, use K as the number of clustering and anchor boxes, the width and height of the central box clustering is used as the dimension of anchor box [19].

The calculation process of K-means method can be summarized as follows:

- Randomly generate K initial positions;
- Classify all data points to the nearest Euclidean center of mass;
- Calculate the position of all data points in a class, and calculate the new center of mass;
- Repeat until the position of the center of mass is no longer changed.

From the k-means thought, it can be seen that its thought based on greedy algorithm is bound to be unable to get the optimal solution Therefore, there are two disadvantages of k-means:

- K should be specified in advance;
- K-means algorithm is very sensitive to the initial position of the initial point.

In this paper, aiming at the randomness of the vehicle target position in the actual scene, K-means++ algorithm is adopted to conduct statistics on the position of anchor box [20]. The calculation method of K-means++ is as follows:

- Randomly select a data point in the data set and set it as the first clustering center;
- Calculate the distance between all data points x and its nearest clustering center D(x);
- Try to update data points. The selection criteria are: points with more data are selected as the new cluster. The probability of heart should be large;
- Repeat until K clustering centers are selected;
- Initialize K clustering centers and run K-means algorithm.

From the k-means ++ algorithm process, it can be seen that k-means++ can select the appropriate clustering center according to the actual application scenario as far as possible to collect data points of the entire dataset, so as to make the vehicle object detection more efficient.

IV. EXPERIMENTS

A. Data Set Preparation

In the experimental section, the model is compared with other typical model in this paper. The algorithm model was trained on the UA-DETRAC [18] data set, the data set is composed of 10 videos, videos was recorded by Cannon EOS 550D camera in 24 different locations in Beijing and Tianjin, China. Videos have 25 frames per second and $960 * 540$ pixels, with a total data set of more than 140000 frames. There were 8,250 vehicles in the data set, and 1.21 million vehicle rectangular boxes were marked. First, the images in the database are sorted out according to the format of VOC2007 data set, and the images in the data set are randomly divided into training set and test set in proportion. Secondly, generate the corresponding target box location information file in XML format. Finally, write python program to normalize the location information of the target box in XML format and convert it into TXT format as the label of vehicle data set.

B. *Experiments Configuration and Training*

This experiment is carried out under the Windows system. Using Intel (R) Core (TM) i7-7700-HQ CPU processor, in order to improve the computing speed and reduce the training time, using the Nvidia Quadro M1200 graphics, CUDA10.0 and call the GPU to accelerate CUD-NN7.5. During the process of training, the indicators of algorithm for dynamic record, along with the increase of the number of iterations, the change trend of average loss function as shown in fig.2.

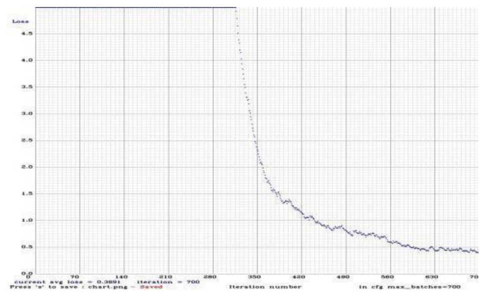


Fig. 2. Network iteration loss value change diagram

C. *Experiments Results*

The experiment is compared with YOLOV3 and SSD methods in terms of average accuracy and average detection time. All three methods were tested on UA-DETRAC test set. We can see from table 1, the mAP of our method in this paper is 79.8% the mAP of SSD method is 71.3% and the mAP is 73.5% in the YOLOV2 method. In the detection of UA-DETRAC data set, it is shown that the average accuracy mean of the method proposed in this paper is higher and more accurate than that based on YOLOV2 model and SDD model. And the average detection time of our method in this paper is 0.117s, SSD method is 0.214s, the average detection time of YOLOV2 method is 0.154s. It shows that our method proposed in this paper has a shorter average detection time than that based on YOLOV2 model and SSD model.

TABLE I
Results of mAP comparison

Method	Datasets	Means Accuracy (mAP)/%	Mean Test Time/s
SSD	UA-DETRAC	71.3	0.214
YOLOV2	UA-DETRAC	73.5	0.154
OUR	UA-DETRAC	79.8	0.117

Test results as shown in figure 3, from the figure (a)(b), you can see that in the case of light conditions is not ideal, our model can keep the recognition accuracy of the vehicle in the figure at a high level. But from figure (c)(d) find that when the distance is too far or there are shielding objects, there will be some missing detection of the vehicle in the figure.

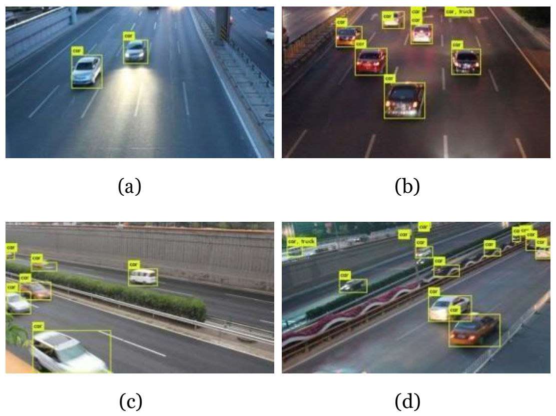


Fig. 3. Test results under different weather and road conditions.

V. CONCLUSION AND FUTURE WORK

In this paper, YOLOV3 model is used for vehicle object detection on the UA-DETRAC vehicle data set. YOLOV3 model effectively avoids the dependence of traditional vehicle object detection model on manual feature selection by using depth convolution feature. Our paper uses label smoothing to reduce the degree of over-fitting in the model, K-means++ clustering to avoid the gradient disappearance problem and improve the training accuracy. Through experiment, our detection model can achieve better vehicle object detection. It can be concluded from the experimental results that our model has a good anti-interference ability to the light when detecting the object. However, it is not robust enough for small targets, and the training network is limited to hardware conditions. The accuracy still needs to be improved, and the performance still needs to be improved, which is what needs to be studied and improved in the later stage.

References

[1] Dalal N, Triggs B. Histograms of Oriented Gradients for Human Detection, Proceedings of the IEEE Computer Society Conference on Computer Vision & Pattern Recognition. San Diego, 2005:886-893.
[2] Lowe D G. Distinctive Image Features from Scale-Invariant Keypoints. International Journal of Computer Vision. 2004,60(2):91-110.
[3] VAPNIK V, CORTES C. Support vector networks. Machine learning, 1995, 20(3):273-297.
[4] Ferreira A J, Figueiredo MAT. Boosting Algorithms: A Review of Theory, Methods, and Application, Ensemble Machine Learning: Methods and Applications. 2012.
[5] Zhou Y,Liu L, Shao L, et al.Fast automatic vehicle annotation for urban traffic surveillance.IEEE Transactions on Intelligent Transportation Systems,2018,19(6):1973-1984.
[6] He K, Zhang X, Ren S,et al. Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition. IEEETrans Pattern Anal Mach Intell, 2014, 37(9):1904-1916.

[7] Ren S, He K, Girshick R, et al. Faster R-CNN: towards real- time object detection with region proposal networks. IEEE transactions.

[8] Dai J, Li Y, He K, et al. R-FCN: Object Detection via Region- based Fully Convolutional Networks. Proceedings of the Neural Information Processing Systems. Barcelona, 2016:379-387.

[9] Redmon J, Divvala S, Girshick R, et al. You Only Look Once: Unified, Real-Time Object Detection. Proceedings of the IEEEConference on Computer Vision and Pattern Recognition. Las Vegas, 2016:779-788.

[10] Redmon J, Farhadi A. YOLOv3: An Incremental Improvement. 2018. arXiv: 1804.02767.

[11] Liu W, Anguelov D, Erhan D, et al. SSD: Single Shot MultiBox Detector.Lecture Notes in Computer Science,2016,9905:21-37.

[12] Redmon J. YOLO: Real-Time Object Detection.[2019-03-29].

[13] Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition. Proceedings of the IEEE In- ternational Conference on Computer Vision. 2014:143-151.

[14] He K, Zhang X, Ren S, et al. Deep Residual Learning for Image Recognition. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. Las Vegas, 2016:770-778.

[15] Szegedy C , Vanhoucke V , Ioffe S , et al. [IEEE 2016 IEEE Con- ference on Computer Vision and Pattern Recognition (CVPR)- Las Vegas, NV, USA (2016.6.27-2016.6.30)] 2016 IEEE Confer- ence on Computer Vision and Pattern Recognition (CVPR)- Rethinking the Inception Architecture for Computer Vision. 2016:2818-2826.

[16] Wen L , Du D, Cai Z, et al. UA-DETRAC: A New Benchmark and Protocol for Multi-Object Detection and Tracking. Com- puter Science, 2015.

[17] Zhang Z, He T, Zhang H, et al. Bag of Freebies for Training Object Detection Neural Networks.2019. arXiv:1902.04103.

[18] Ren ZJ, Lin SZ, Li DW, et al. Mask R-CNN Object Detection Method Based on Improved Feature Pyramid. Laser & Opto- electronics Progress, 2019, 56(4): 41502

[19] Park Ji-Hoon,Hwang Hye-Won,Moon Jun-Ho,Yu Young-sung,Kim Hansuk,Her Soo-Bok,Srinivasan Girish,Aljanabi Mohammed Noori A,Donatelli Richard E,Lee Shin-Jae. Automated identification of cephalometric landmarks: Part 1-Comparisons between the latest deep-learning methods YOLOV3 and SSD. The Angle orthodontist,2019.

[20] Zhang Z, He T, Zhang H, et al. Bag of Freebies for Training Object Detection Neural Networks.2019. arXiv:1902.04103.

[21] IOFFE S,SZEGEDY C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. [2015-03-02].