HIDDEN OBJECT IDENTIFICATION FOR AUTONOMOUS CAR USING CNN

Submitted in partial fulfillment of the

requirements for the award of

Bachelor of Engineering degree in Computer Science and Engineering

By

PRAKASH M (Reg.No - 39110787)
JANARTHANAN M (Reg.No - 39110396)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SCHOOL OF COMPUTING

SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited with Grade "A" by NAAC | 12B Status by UGC | Approved by AICTE
JEPPIAAR NAGAR, RAJIV GANDHISALAI,
CHENNAI - 600119

APRIL - 2023

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited with Grade "A" by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of **PRAKASH M (Reg. No - 39110787)** and **JANARTHANAN M (Reg.No - 39110396)** who carried out the Project Phase-2 entitled **"HIDDEN OBJECT IDENTIFICATION FOR AUTONOMOUS CAR USING CNN"** under my supervision from December 2022 to April 2023.

Internal Guide

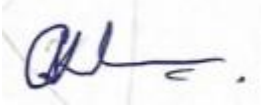
Ms.D. DEVI, M.E

Head of the Department

Dr. L. LAKSHMANAN, M.E., Ph.D.

Submitted for Viva voce Examination held on 24.04.23

InternalExaminer



ExternalExaminer

DECLARATION

I, **PRAKASH M (Reg.No - 39110787),** hereby declare that the Project Phase-2 Report entitled **HIDDEN OBJECTS IDENDIFICATION FOR AUTONOMOUS CAR USING CNN** done by me under the guidance of **MS.D.DEVI, M.E** is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering degree in **Computer Science and Engineering**.

DATE: 24.04.23
PLACE:Chennai

SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to **Board of Management** of **SATHYABAMA** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T. Sasikala M.E., Ph. D**, **Dean**, School of Computing, **Dr. L. Lakshmanan M.E., Ph.D.,** Head of the Department of Computer Science and Engineering for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Ms.D.Devi M.E,** for her valuable guidance, suggestions and constant encouragement paved way for the successful completion of my phase-2 project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the **Department of Computer Science and Engineering** who were helpful in many ways for the completion of the project.

ABSTARCT

Hidden object identification / detection is a vast, vibrant and complex area of computer vision. If there is a single object to be detected in an image, it is known as Image Localization and if there are multiple objects in an image, then it is Object Detection. This detects the semantic objects of a class in digital images and videos. One of the most anticipated 21st-century technologies and one of the subjects of the current study that is gaining the greatest attention is autonomous driving. By detecting and responding to the vehicle's immediate environment, autonomous driving tries to navigate roads without human assistance. It presents significant difficulties for computer vision and machine learning. Processing several candidate object locations, which are frequently referred to as "proposals," is one difficulty. These choices only offer basic localization, which needs to be refined in order to obtain precise localization. However, solutions to these issues frequently come at the expense of efficiency, precision, or simplicity. The detection of objects around an autonomous vehicle is essential to operate safely. This paper presents an algorithm to detect objects around an autonomous car. All objects are classified as moving or stationary as well as by type (e.g. vehicle, pedestrian, biker or other). **Convolution Neural Networks** and **YOLO** are the representative tools of **Deep learning** is implemented to detect objects using OpenCV (Opensource Computer Vision), PyTorch and Detectron 2 which is a library of programming functions mainly aimed at real time computer vision.

TABLE OF CONTENTS

LIST OF ABBREVIATION

CNN - Convolutional Neural Networks

YOLO - You Only Look Once

OpenCV - Open Source Computer Vision

Lidar - Light imaging, detection and ranging

GPS - Global Positioning System

IEEE - Institute of Electrical and Electronics Engineers

CNN - Convolutional Neural Networks

R CNN - Regions Convolutional Neural Networks

CF - Car Following

ATAC - Adjustment on Tsallis Actor-Critic

NGSIM - Next Generation Simulation

RL - Reinforcement Learning

DNNs - Deep Neural Networks

ASA - Adaptive Square Attack

ES - Energy Strategy

ECU - Electronic Control Unit

AVs - Autonomous Vehicle

YOLOv3 - You Look Once version 3

RL - Reinforcement Learning

AMoD - Autonomous Mobility-on-Demand

VKT - Vehicle kilometers travelled

FNR - False Negative rate

FPR - False Positive rate

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

In recent years, autonomous/self-driving cars have drawn much interest as a topic of research for both academia and industry. For a car to be a truly autonomous, it must make sense of the environment through which it is driving. The autonomous car must be able to both localize itself in an environment and identify and keep track of objects (moving and stationary). The car gets information about the environment using exteroceptive sensors such Lidar, cameras, inertial sensors, and GPS. The information from these sensors can be used together and fused to localize the car and track objects in its environment, allowing it to travel successfully from one point to another.

The process of path planning and autonomous vehicle guidance depends on three things: localization, mapping, and tracking objects. Localization is the process of identifying the position of the autonomous vehicle in the environment. Mapping includes being able to make the sense of the environment. Tracking of moving objects involves being able to identify the moving objects and track them during navigation.

The motive of object detection is to recognize and locate all known objects in a scene. Preferably in 3D space, recovering pose of objects in 3D is very important for autonomous Cars. Object detection is relatively simpler if the machine is looking for detecting one particular object. However, recognizing all the objects inherently requires the skill to differentiate one object from the other, though they may be of same type. Such problem is very difficult for machines, if they do not known about the various possibilities of objects.

CHAPTER 2

LITERATURE SURVEY

This problem statement has been extensively studied over the past 5 years by researchers and automotive companies in a bid to create a solution, and all their solutions vary from analyzing various algorithms and methods to detecting objects around the autonomous cars.

2.1 INFERENCES FROM LITREATURE SURVEY

In order to understand the development of research in autonomous driving in the last years, itis important to conduct a literature review to understand the different fields of application through which autonomous driving has evolved as well as to identify research gaps.

According to a survey with more than 200 experts on autonomous vehicles by the IEEE(2014), the world's largest professional association for the advancement of technology, the three biggest obstacles to reach the mass adoption of driverless cars are: legal liability, policymakers and customer acceptance, while the following three; cost, infrastructure and technology are seen as less of a problem.

In the next we can see through a literature review how the development of research has focused mainly on technology development and just begun to focus on legal liability and policymakers while research on customer acceptance has been more limited.

Autonomous cars are not that far away, for example Audi and Mercedes (Bartl, 2013) have announced almost being ready from production in highly automated features. As a reflection of the daily news, we can steadily see how this technology manages to get closer to be in our everyday life, with examples of cars driving a blind man for tacos already on 2012 (Google, 2012), coast to coast trips (CNN, 2015), an Italy to China trip (Broggi, et al., 2013) and 700,000 miles already travelled by Google (Urmson , 2014).

But main automakers in the race such as Audi, BMW, Cadillac, Ford, General Motors, Jaguar, Land Rover, Lincon, Mercedes-Benz, Nissan, Tesla and Volvo, are trying to integrate it slowly to their models despite the fairly readiness of the technology. This can be interpreted as a futile attempt to keep this totally disruptive technology under control and to have overall slower customer integration, but this old model will prove to be not good enough due to the magnitude and impact of this technology (Bartl, 2015).

2.2 OPEN PROBLEMS IN EXISTING SYSTEM

Deep learning is one of the main technologies that enabled self-driving. It's a versatile tool that can solve almost any problem – it can be used in physics, for example, the proton-proton collision in the Large Hadron Collider, just as well as in Google Lens to classify pictures. Deep learning is a technology that can help solve almost any type of science or engineering problem., we'll focus on deep learning algorithms in self-driving cars – convolutional neural networks (CNN). CNN is the primary algorithm that these systems use to recognize and classify different parts of the road, and to make appropriate decisions. Along the way, we'll see how Tesla, Waymo, and Nvidia use CNN algorithms to make their cars driverless or autonomous.

In this paper [1], a machine learning algorithm is proposed. This algorithm is mostly used to develop shape models and aids in lane detection and traffic sign detection by detecting shapes. Both projects are written in Python and use the OpenCV2 and NumPy libraries as well as the Hough Detection technique to identify the proper traffic light circles. All of these tools are used to train all shape models using supervised training algorithms, and the detection is carried out in a way that enables autonomous vehicles to recognize lane markings and traffic signs.

In order to improve detection accuracy, this article [2] fuses Faster R-CNN and YOLO v2 using the Kalman filter. Due to its greater accuracy, the findings from Faster R-CNN are used as observations, while those from YOLO v2 are used as state variables. The experiment uses video samples with pictures of vehicles. The outcomes demonstrate that the Kalman filter can improve object detection when two methods are combined.

The goal of this paper [3], is to conduct a comparative analysis of the YOLO model, which is used in object detection and is intended to aid the visually disabled in navigating their environment. Detecting instances of specific classes of things from an image is done through object detection. Recent years have seen a swift transformation in object detection, giving rise to many sophisticated and challenging algorithms like YOLO, SSD, Fast R-CNN, Faster R-CNN, HOG, and many others. The YOLO algorithm's architecture, which is frequently used in object detection and object classification, is explained in this study paper. They trained the model using the COCO dataset. The goal of this research project is to find the best way to apply the YOLO model.

This paper [4], is divided into two sections. A car, a person, a truck, a bus, a traffic light, a motorcycle, a pothole, and a wetland are just a few examples of the classes of objects that are implemented in the first section. Yolo (You Only Look Once)Algorithm is used to detect objects in the environment, and it provides precise classification and position that is configured on newly created datasets for these classes of objects. The autonomous vehicle can travel without getting stuck in potholes by detecting them on Indian roads. The suitability of the running objects is investigated in part two of the suggested method, which is implemented on the Raspberry Pi4, a well-known embedded computer board. That improves the effect on object detection and solves problems in the real world.

In this paper [5], two models are created to recognize traffic lights in videos and pictures. Data gathered from Bahraini roadways is put into a dataset. To present the best model, the two are assessed and contrasted. While the second model employs YOLO algorithm weights and the OpenCV library to analyse video frames, the first model is founded on the analysis of still images. In comparison to the first model, which analyses still images, it was discovered that the second model was capable of accurately detecting all traffic signals in video frames.

In this paper [6], a deep neural network was trained to recognize and categorise traffic lights. Additionally, the output of this model was given to an algorithm that would notify the driver in the event that the computer detected that the driver might

cross the red light; this feature could be incorporated into current Advanced Driver Assistance Systems (ADAS) systems in a car. A camera-based approach was also suggested to remove pointless traffic lights from a scenario.

In this study [7], many self-driving automobiles that are aware of their surroundings use LiDAR (Light Detection and Ranging) sensors, which distorts the data that is collected. These meteorological conditions make self-driving cars less safe. The performance of 3D object detection suffers as a result of the point clouds' missing points issues, which are also a result of the harsh weather. In order to recover missing points brought on by snow, we therefore suggest a novel probability estimation technique and a Miss-Convolution layer. Models that perform well under typical settings are outperformed by the proposed work. In conclusion, snow frequently leads to detection errors for 3D contemporary detectors. We considerably improve the 3D detector's performance in snowy weather by recovering missing points from the point cloud.

In this article [8], they focus on the 3D object localization and detection task of an autonomous vehicle control system. They examine current detection techniques that draw on heterogeneous data from a variety of instruments, including cameras, depth sensors, and LIDARs. They perform experimental studies using real and simulated data obtained using CARLA, an open-source simulator for autonomous driving study, to assess the efficacy of these methods in an autonomous driving problem.

In this research [9], the Kalman filter—one of the traditional object tracking techniques—was combined with Tiny YOLOv3, one of the most effective object detection architectures. It is suggested to use a quick and compact object detection method that improves the model's precision without slowing it down.

The goal of the research [10], is to identify a practical method for identifying objects in images from the self-driving vehicle medium. In this study, deep convolution networks are used to precisely identify objects. They used the BDD100K dataset, one of the largest open-source datasets for autonomous driving released by Berkeley University, to train and evaluate the neural network. The proposed algorithm takes the strategy of applying the feature pyramid network along with a single-stage object

detector, which improves object detection accuracy.

This research [11], develops a universal deep imitation reinforcement learning system (DIRL) that successfully provides nimble autonomous racing with visual inputs. Both IL and model-based RL are used to learn driving skills, allowing the agent to securely interact with an offline world model while still learning from human instructors. This approach is tested on a real-world 1/20-scale RC car with minimal onboard processing power as well as in a high-fidelity driving simulation. The evaluation findings show that the method performs better in terms of sample efficiency and task performance than earlier IL and RL methods.

In this paper [12], they suggest the prediction of steering angle and speed control in autonomous vehicles based on the detection of objects in street view, such as cars in front, traffic signs, pedestrians, and lane lines. the method of object detection, steering angle prediction, and speed control prediction using video recorded with a single camera and a convolutional neural network (CNN). This technique eliminates the need for additional sensors like LIDAR and RADAR, which lowers costs and addresses the drawbacks of those sensors.

Carpooling and the ride-sharing [13] idea are currently resolving many issues faced by modern societies. The issues regarding the overuse of oil, traffic jams, inefficient use of time, pollution due to overuse of vehicles on the road, and health problems. It is also expected that ride-sharing and carpooling will be more efficient for autonomous vehicles because of their unmanned nature and full-fledged autonomy. When unmanned cars will do the responsibility of carpooling and ride-sharing or car-hailing, many issues regarding booking rides, location sharing, payment handling, and privacy issues must be improved. To cope with these issues, mainly concerning the scheduling of resources, we need effective scheduling techniques to handle all kinds of emotional problems and provide a pollution-free and accident-free environment on autonomous vehicles' roads. Among other approaches, we feel that Data Science provides a perfect opportunity to leverage machine learning models to classify and see what parameters can encourage people to opt for a move towards connected autonomous vehicles. In this paper, we discuss autonomous vehicles, Vehicle-as-a-Service, and their role in reducing CO2 emissions. The dataset used in

this study gives insights into the city of Chicago's taxi trips. The dataset includes data about taxi trips, their respective duration, and anonymized data about the passengers. We also discuss some studies by a taxonomy that will identify the gap of an optimal incentive mechanism that will influence users to join carpooling in autonomous cars instead of having their vehicles.

In this paper [14], automated object identification and navigation decision modules were used to build a car-following framework for AVs. The goal is to make it possible for an AV to follow another vehicle using RGB-D (Red, Green, and Blue Depth) frames. They suggested using a joint approach that combined a reinforcement learning (RL) algorithm for guiding the self-driving car with the You Look Once version 3 (YOLOv3) object detector to identify the leader vehicle and other obstacles. Q-learning and Deep Q-learning, two RL algorithms, have been researched. The results of the simulations demonstrate the convergence of the created models and examine their effectiveness in following the leader. They demonstrated that AVs can adopt a suitable car-following behaviour and that promising results can be obtained using simply video frames.

Car-following theory [15] has received considerable attention as a core component of Intelligent Transportation Systems. However, its application to the emerging autonomous vehicles (AVs) remains an unexplored research area. AVs are designed to provide convenient and safe driving by avoiding accidents caused by human errors. They require advanced levels of recognition of other drivers' driving-style. With car-following models, AVs can use their built-in technology to understand the environment surrounding them and make real-time decisions to follow other vehicles. In this paper, we design an end-to-end car-following framework for AVs using automated object detection and navigation decision modules. The objective is to allow an AV to follow another vehicle based on Red Green Blue Depth (RGB-D) frames. We propose to employ a joint solution involving the You Look Once version 3 (YOLOv3) object detector to identify the leader vehicle and other obstacles and a reinforcement learning (RL) algorithm to navigate the self-driving vehicle. Two RL algorithms, namely Q-learning and Deep Q-learning have been investigated. Simulation results show the convergence of the developed models and investigate their efficiency in following the leader. It is shown that, with video frames only,

promising results are achieved and that AVs can adopt a reasonable car-following behaviour.

Increasing automation [16] in road vehicles moves decision making from humans to algorithms. This gives a lack of clarity in assigning responsibility for the consequences of the resultant actions, presenting problems for ethicists, lawyers, and engineers. This article applies proven methods based on the 4D/RCS reference model architecture to derive proposed architectures for autonomous vehicles with J3016 autonomy levels 2-5. The architectures have a hierarchy of nodes, which can be either a human or an automated process, giving unambiguous allocation of authority to act and consequent responsibilities to individual nodes in the system. A concept called authorized power is introduced for every node, giving clear limits on each node's ability to act. Setting node requirements gives a firm basis for engineers and lawyers to agree where responsibilities lie, whether with the user, owner, or in the supply chain. It also provides a framework to assess engineering risk and the wider financial implications of the new technologies as they are introduced.

To better understand [17] the road condition and make correct driving decisions, traffic sign recognition becomes a crucial component commonly equipped in the vision system of modern autonomous cars. The state-of-the-art traffic sign recognition models are designed with the backbones of deep neural networks (DNNs) since DNNs are powerful to extract more effective visual features that benefit recognition performance. As the recent studies on adversarial attacks have shown that DNNs are easy to be fooled by perturbed images and lead to misclassification, in this article, we explore the vulnerability of the DNN-based traffic sign recognition model. Most existing adversarial attack methods limitedly focus on the white-box attack on the recognition models whose underlying configurations (e.g., network architectures and parameters) are accessible. Differently, we propose a novel attacking method dubbed adaptive square attack (ASA) that can accomplish the black-box attack, i.e., bypassing the access the configurations of the recognition models. Specifically, the proposed ASA method employs an efficient sampling strategy that can generate perturbations for traffic sign images with fewer query times. Extensive experiments on the benchmark data set German traffic sign recognition benchmark with large-scale traffic sign images for autonomous cars show

that our proposed ASA method is advanced to perform the black-box attack with high efficiency. Although the generated adversarial traffic sign images by the proposed ASA method are visually similar to the raw images with almost imperceptible differences, they can successfully lead to the misclassification of the state-of-the-art recognition model.

This paper [18], makes use of the TensorFlow object detection API to identify moving things. The object tracking algorithm is also given the location of the object that was identified. For reliable object identification, a brand-new CNN-based object tracking algorithm is employed. The suggested method can find the item in various lighting and occlusion conditions.C

CHAPTER 3

REQUIREMENT ANALYSIS

This project's main objective is to detect multiple objects around an autonomous car using deep learning algorithm models. Essential requirements of the system and its design constraints are discussed below.

3.1 REQUIREMENTS

3.1.1 Functional Requirements

A deep learning model is required to perform the object detection task over the images captured at the intersections. There is a need of pre-trained model weights in-order to reduce the training time of the model. An algorithm must be devised to process the data from the model and detect the objects for the autonomous car. All these are expected to be processed in real-time.

- **CNN:** A convolutional neural network (CNN) is a feed-forward neural network that is generally used to analyze visual images by processing data with a grid-like topology. It's also known as ConvNet. A convolutional neural network is used to detect and classify objects in an image.

- **R-CNN:** R-CNN stands for Region-based Convolutional Neural Network. The key concept behind the R-CNN series is region proposals. Region proposals are used to localize objects within an image. In the following blogs, I decided to write about different approaches and architectures used in Object Detection.
- **YOLO:** YOLO or You Only Look Once, is a popular real-time object detection algorithm. YOLO combines what was once a multi-step process, using a single neural network to perform both classification and prediction of bounding boxes for detected objects.

3.1.2 Interface Requirements

The cameras present at the intersections of car must be able to capture images of the view of the road at high quality. This is essential so that the model is able to make out the type of objects are present with higher accuracy. Need of CPU and GPU to carry out computations.

3.1.3 Performance Requirements

The algorithm must be fast and efficient in processing the data. The deep learning algorithm must be accurate in identifying different types of objects which are in the road.

3.1.4 Hardware Requirement for Implementation

- **Processor:** Intel Core i5 7th Gen or More.
- **System Architecture:** Windows 64 bit x86 or 32-bit x86 or MacOS 64-bit x 86.
- **Network**: Ethernet Connection(LAN) or a Wireless adapter(Wi-Fi).
- **Hard Disk:** Minimum 120 GB; Recommended 1TB or more.
- **Memory (RAM):** Minimum 2GB; Recommended 4GB or above.

3.1.5 Software Requirement for Implementation

- **Python IDE:** Any Python Editors (Version - 3.7.2) like PyCharm, IDLE, etc.
- **Operating system :** Windows 10 or older or MacOS 10.13 + or Linux.
- **GUI :** Google Colab or Jupyter Notebook or Kaggle Notebook editor.
- PyTorch

3.2 TOOLS AND TECHNOLOGIES USED:

3.2.1 Python

Python is a multi-paradigm programming language. Object-oriented programming and structured programming are fully supported, and many of their features support functional programming and aspect-oriented programming (including metaprogramming and metaobjects). Many other paradigms are supported via extensions, including design by contract and logic programming.

Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly procedural), object-oriented and functional programming. It is often described as a "batteries included" language due to its comprehensive standard library.

Fig 3.1: Python

Python was conceived in the late 1980s by Guido van Rossum at Centrum Wiskunde & Informatica (CWI) in the Netherlands as a successor to the ABC programming language, which was inspired by SETL, capable of exception handling and interfacing with the Amoeba operating system. Its implementation began in December 1989. Van Rossum shouldered sole responsibility for the project, as the lead developer, until 12 July 2018, when he announced his "permanent vacation" from his responsibilities as Python's "benevolent dictator for life", a title the Python community bestowed upon him to reflect his long-term commitment as the project's chief decision-maker. In January 2019, active Python core developers elected a five-member Steering Council to lead the project.

3.2.2 PyTorch

PyTorch is a machine learning framework based on the Torch library, used for applications such as computer vision and natural language processing, originally developed by Meta AI and now part of the Linux Foundation umbrella. It is free and open-source software released under the modified BSD license. Although the Python interface is more polished and the primary focus of development, PyTorch also has a C++ interface.

A number of pieces of deep learning software are built on top of PyTorch, including Tesla Autopilot, Uber's Pyro, Hugging Face's Transformers, PyTorch Lightning, and Catalyst.

PyTorch provides two high-level features:

- Tensor computing (like NumPy) with strong acceleration via graphics processing units (GPU)
- Deep neural networks built on a tape-based automatic differentiation system

Fig 3.2: PyTorch

Meta (formerly known as Facebook) operates both *PyTorch* and *Convolutional Architecture for Fast Feature Embedding* (Caffe2), but models defined by the two frameworks were mutually incompatible. The Open Neural Network Exchange (ONNX) project was created by Meta and Microsoft in September 2017 for converting models between frameworks. Caffe2 was merged into PyTorch at the end of March 2018. In September 2022, Meta announced that *PyTorch* would be governed by PyTorch Foundation, a newly created independent organization – a subsidiary of Linux Foundation.

3.2.3 Git

Git is a distributed version control system that tracks changes in any set of computer files, usually used for coordinating work among programmers collaboratively developing source code during software development. Its goals include speed, data integrity, and support for distributed, non-linear workflows (thousands of parallel branches running on different systems).

Git was originally authored by Linus Torvalds in 2005 for development of the Linux kernel, with other kernel developers contributing to its initial development. Since 2005, Junio Hamano has been the core maintainer.

As with most other distributed version control systems, and unlike most client–server systems, every Git directory on every computer is a full-fledged repository with complete history and full version-tracking abilities, independent of network access or a central server. Git is free and open-source software distributed under the GPL-2.0-only license.

Fig 3.3: Git

Git development began in April 2005, after many developers of the Linux kernel gave up access to BitKeeper, a proprietary source-control management (SCM) system that they had been using to maintain the project since 2002. The copyright holder of BitKeeper, Larry McVoy, had withdrawn free use of the product after claiming that Andrew Tridgell had created SourcePuller by reverse engineering the BitKeeper protocols. The same incident also spurred the creation of another version-control system, Mercurial.

The development of Git began on 3 April 2005. Torvalds announced the project on 6 April and became self-hosting the next day. The first merge of multiple branches took place on 18 April. Torvalds achieved his performance goals; on 29 April, the nascent

Git was benchmarked recording patches to the Linux kernel tree at the rate of 6.7 patches per second. On 16 June, Git managed the kernel 2.6.12 release.

Torvalds turned over maintenance on 26 July 2005 to Junio Hamano, a major contributor to the project. Hamano was responsible for the 1.0 release on 21 December 2005.

CHAPTER 4

DESCRIPTION OF PROPOSED SYSTEM

Self Driving cars in the existing work [1] operate through sensors, actuators, and an embedded system control. In this case, lane keeping is crucial for safety precautions to stop traffic accidents. This research suggests a machine learning algorithm for determining the optimal form. The training model makes it feasible to identify this. The suggested algorithm uses the Hough line transformation method to find any shape.

4.1 PROPSED SYSTEM

In our system, the objects close to the autonomous vehicle are identified using the CNN and YOLO algorithm. The dataset Images can be trained using convolutional neural networks (CNN) and YOLO networks with the help of the library, which are used to detect the objects. These algorithms are employed to automatically detect thousands of pedestrians, bikers, cars, trucks and traffic lights both in images and videos.

CNN used for self-driving cars

Convolutional neural networks (CNN) are used to model spatial information, such as images. CNNs are very good at extracting features from images, and they’re often seen as universal non-linear function approximators.

CNNs can capture different patterns as the depth of the network increases. For example, the layers at the beginning of the network will capture edges, while the deep layers will capture more complex features like the shape of the objects (leaves in trees, or tires on a vehicle). This is the reason why CNNs are the main algorithm in self-driving cars.

The key component of the CNN is the convolutional layer itself. It has a convolutional kernel which is often called the filter matrix.

The dimension of the filter matrix in practice is usually 3X3 or 5X5. During the training process, the filter matrix will constantly update itself to get a reasonable weight. One of the properties of CNN is that the weights are shareable. The same weight

parameters can be used to represent two different transformations in the network. The shared parameter saves a lot of processing space; they can produce more diverse feature representations learned by the network.

The output of the CNN is usually fed to a nonlinear **activation function**. The activation function enables the network to solve the linear inseparable problems, and these functions can represent high-dimensional manifolds in lower-dimensional manifolds. Commonly used activation functions are Sigmoid, Tanh, and ReLU, which are listed.

4.2 PROCESS MODEL

- **Dataset preparation:** Collect and preprocess the data. This includes gathering images that contain the objects of interest, labeling the objects with bounding boxes, and splitting the data into training, validation, and test sets.

- **Training:** Train a CNN model on the labeled training data. This involves feeding the training images through the CNN and adjusting the model's parameters to minimize the difference between the predicted and actual bounding boxes.

- **Validation:** Validate the performance of the trained model on the validation set. This involves evaluating the model's ability to correctly predict the bounding boxes of objects in images that it has not seen during training.

- **Testing:** Test the final performance of the model on the test set. This involves evaluating the model's ability to generalize to new, unseen data.

- **Post-processing:** Apply post-processing techniques to the output of the model to refine the predicted bounding boxes and reduce false positives. This includes techniques like non-maximum suppression (NMS), which removes overlapping bounding boxes and keeps only the most confident predictions.

- **Visualization:** Visualize the final output of the model on new images to evaluate its performance and identify areas for improvement.

Overall, object detection using CNNs involves a combination of data preparation, model training, validation, testing, post-processing, and visualization steps to create an accurate and reliable object detection system.

4.3 ARCHITECTURE DESIGN OF PROPOSED SYSTEM

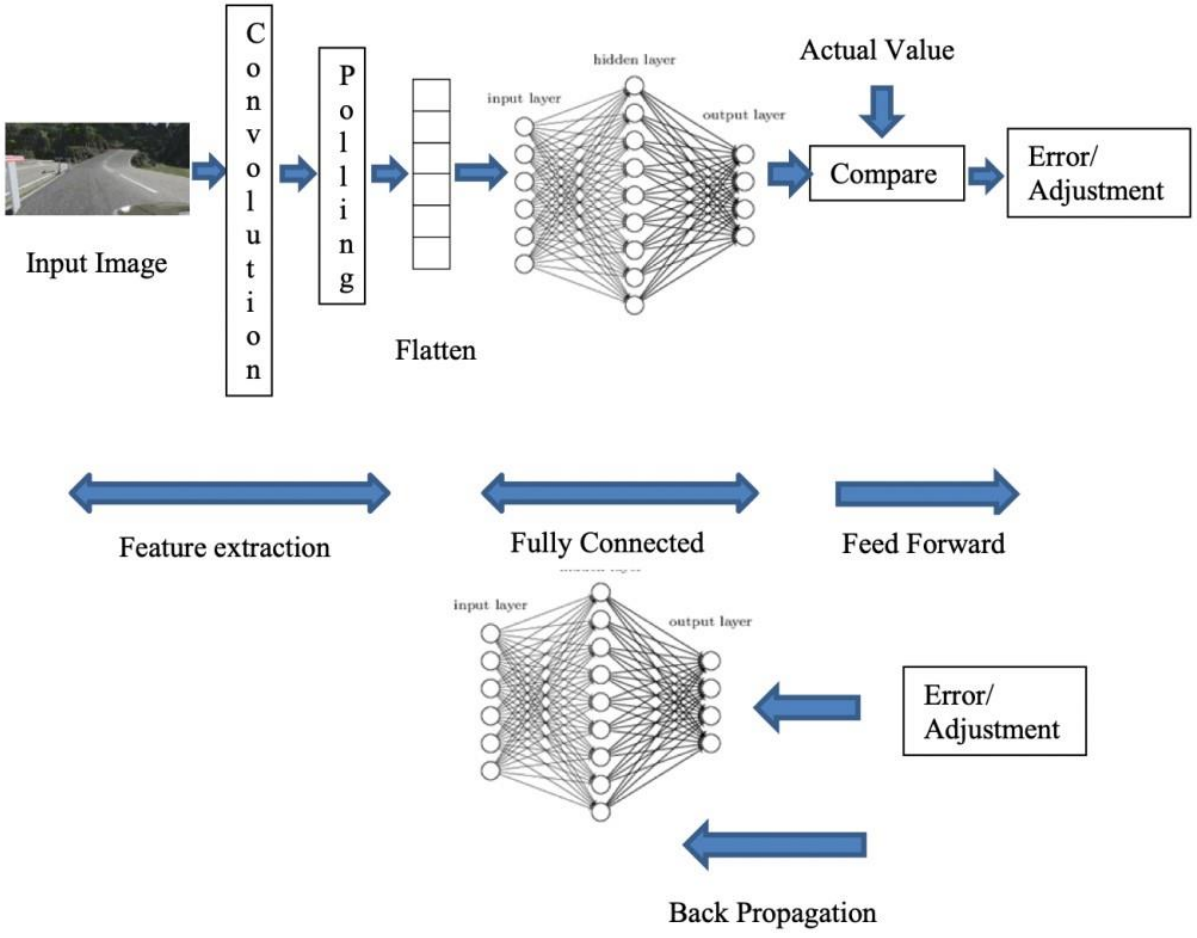


Fig 4.3: System Architecture of CNN

4.4 TESTING PLAN OF THE PROPOSED MODEL/SYSTEM

TensorBoard is a visualization tool for deep learning experiments that is part of the TensorFlow framework. It will monitor and visualize various aspects of our deep learning model, such as the loss, accuracy, and other metrics during training, as well

as the model architecture and performance. It can also be used to compare the performance of different models or hyperparameters, by overlaying the visualizations of multiple runs on top of each other. This allows us to easily compare and analyze the performance of different models or hyperparameters, and make informed decisions about how to improve our deep learning model.

To evaluate our CNN and YOLO models using TensorBoard, we can use the Precision-Recall curve, which is a commonly used metric for object detection performance.

1. **Run inference on the test set:** Use the trained object detection model to run inference on the test set. For each image in the test set, the model should produce a set of bounding boxes and associated class labels and confidence scores.
2. **Compute precision and recall:** For each class, compute the precision and recall values at various confidence score thresholds. Precision is the fraction of detected objects that are actually of the correct class, and recall is the fraction of all objects of the correct class that are detected. The precision-recall curve plots precision versus recall for varying confidence score thresholds.
3. **Log the precision and recall values:** Use TensorFlow's summary operations to log the precision and recall values for each class at various confidence score thresholds to TensorBoard.
4. **Visualize the precision-recall curve:** Open the TensorBoard web interface and navigate to the PR Curve dashboard. Choose the class that to evaluate and view the precision-recall curve for that class. Adjust the confidence score threshold to explore the trade-off between precision and recall. A good object detection model should have high precision and recall at high confidence score thresholds.
5. **Adjust the model and re-evaluate:** If the model performance is not satisfactory, adjust the model hyperparameters or architecture and re-run the inference and logging steps to obtain new precision and recall values. Repeat this process until you are satisfied with the model's performance

By using TensorBoard to visualize the precision-recall curve, we can quickly evaluate the performance of your object detection models and make informed decisions about how to improve it.

The **Precision** is the ratio of true positives over the sum of false positives and true negatives. It is also known as positive predictive value. Precision is a useful metric and shows that out of those predicted as positive, how accurate the prediction was.

Recall is the ratio of correctly predicted outcomes to all predictions. It is also known as sensitivity or specificity.

Accuracy is the ratio of correct predictions out of all predictions made by an algorithm. It can be calculated by dividing precision by recall or as 1 minus false negative rate (FNR) divided by false positive rate (FPR).

The **F1-score** combines these three metrics into one single metric that ranges from 0 to 1 and it takes into account both Precision and Recall.

CHAPTER 5

IMPLEMENTATION DETAILS

5.1 DEVELOPMENT AND DEPLOYMENT SETUP

5.1.1 Datasets Preparation

Object Identification requires more datasets than traditional image classifiers. The development of image recognition algorithms for autonomous driving is implemented here by using the Udacity Self driving car dataset obtained from the Roboflow site - https://public.roboflow.com/object-detection/self-driving-car.

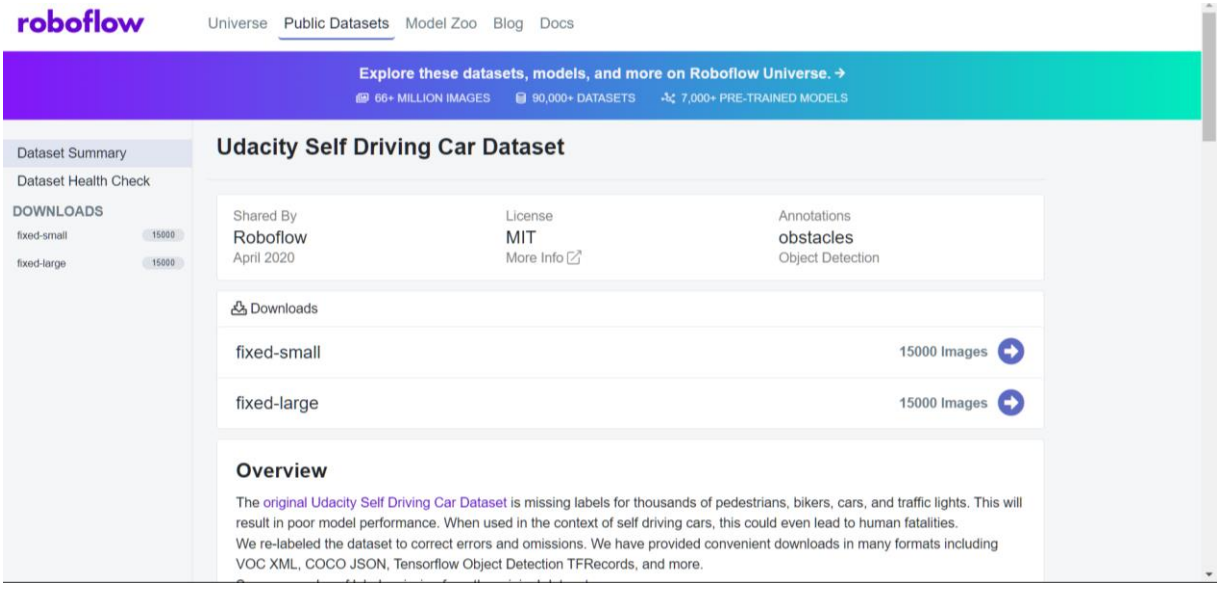


Fig 5.1: Udacity Self Driving Car Dataset

The dataset contains 30,000 number of photos and 97,942 labels across 11 different classes such as car, pedestrains, trucks, bikers, traffic lights and traffic signals. All images are 1920x1200 pixels in size and are appropriate for most common machine learning models. (including YOLO v3, Mask R-CNN, SSD, and mobilenet).

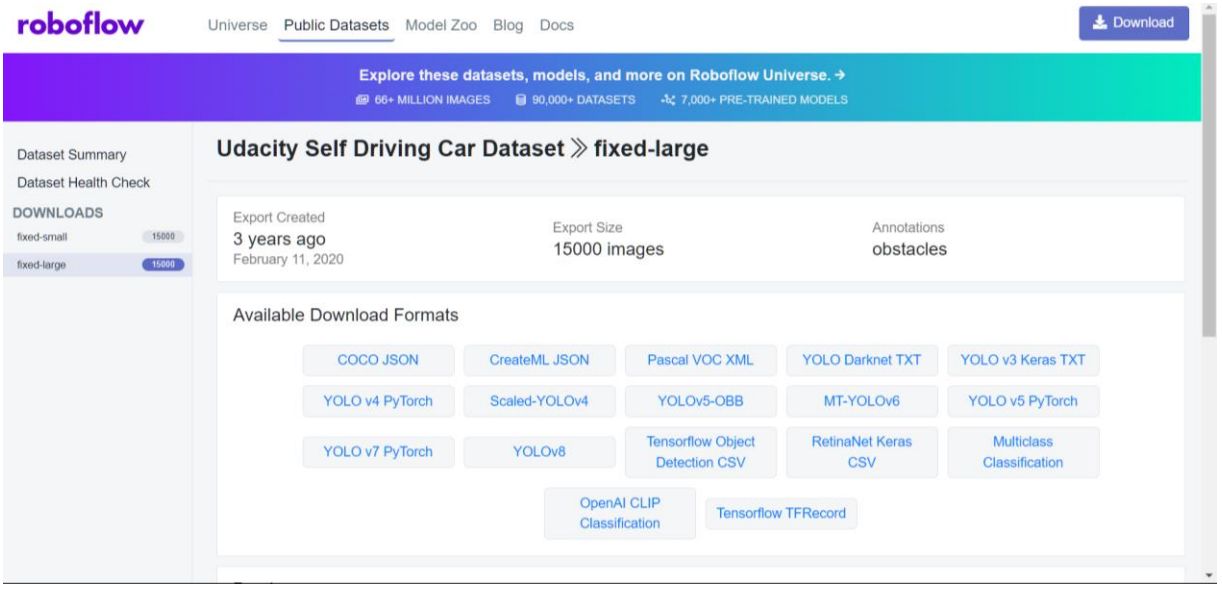


Fig 5.2: Dataset Formats

This dataset is available for download in a variety of forms, including VOC XML, COCO JSON, Tensorflow Object Detection TFRecords, and more. We downloaded the dataset in YOLOv5 Pytorch and COCO JSON file for our purposes. To address particular detection issues, a number of bounding box-labelled datasets were produced. They serve as benchmarks for contrasting various designs and algorithms and for establishing objectives for solutions.

5.1.2 Data Preprocessing

Preprocessing is an important step in deep learning for object detection because it can improve the accuracy and efficiency of the model. A dataset must be preprocessed to a new raw data structure that is network-compatible.

- **Data normalization:** Preprocessing can involve normalizing the data to ensure that the input data has similar ranges of values. This is important because it can help to reduce the impact of differences in scale, which can affect the performance of the model.

- **Data augmentation**: Preprocessing can involve data augmentation techniques such as flipping, rotating, or adding noise to the images. This can help to increase the size of the dataset and improve the robustness of the model.

- **Image resizing:** Preprocessing can involve resizing the images to a uniform size. This is important because it can help to reduce the computational cost of processing the images and improve the efficiency of the model.

- **Object annotation:** Preprocessing can involve annotating the images with the location and size of the objects that need to be detected. This is important because it provides the ground truth labels that the model can learn from.

By performing preprocessing on the dataset, a data can be prepared in a format that can be used to train an object detection model. This can help to improve the accuracy and efficiency of the model when it is deployed for inference on new, unseen data.

Fig 5.3: Data Preprocessing for YOLO Model

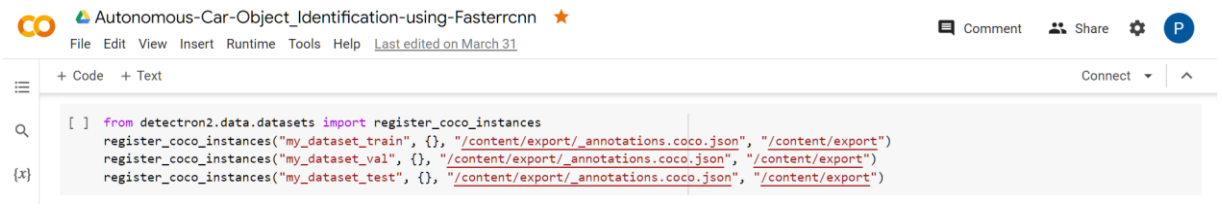


Fig 5.4: Data Preprocessing for Faster R-CNN Model

All files in the dataset are extracted using the format code downloaded from the roboflow site and preprocessed by splitting the data for training, testing, and validating reasons, where their associated images and labels can be stored (Fig

5.1.2.1, Fig 5.1.2.2). Splitting the data into training, testing, and validation sets is an essential step in deep learning to ensure that your model generalises well to new, unknown data. We can train, test, and verify our custom YOLO and CNN model by storing the associated images and labels for each split. We need to keep track of which images and labels are in each split so that the performance of our model can be properly assessed. Everything that our deep learning models want to identify must be annotated, labeled, and delimited by bounding boxes first.

5.1.3 Object identification Models

The choice between YOLO and Faster R-CNN model based object detectors depends on the specific application's requirements.

1. YOLO Model is specifically designed for object detection, where the goal is to identify and locate objects within an image. YOLO is a one-stage object detector that is known for its speed and efficiency, as it can detect objects in real-time. In evaluation, YOLO is often evaluated based on its precision, recall, and average precision (AP), which are measures of how well it detects objects in an image.
2. Faster R-CNN model are generally used for tasks such as image classification and segmentation, where the goal is to classify the entire image or segment it into different regions. CNNs can be very effective at these tasks because they are able to learn complex features and patterns from the input images. In evaluation, CNNs are often evaluated based on their accuracy, precision, recall, and F1-score, which are measures of how well they classify images or segments.

The evaluation metrics used for each model will depend on the task and the specific requirements of the application. If speed is critical for real-time processing, then a YOLO detector may be preferred. However, if accuracy is the priority, then a Faster R-CNN detector may be more suitable.

5.2 Algorithms

5.2.1 Training in YOLO Algorithm

The YOLO model network architecture includes 24 convolutional layers, which are followed by data pooling layers and completely connected layers (Fig.5.2.1.1). These convolutional layers are designed to learn and extract various features from the input image at different levels of abstraction. At various levels of abstraction, these convolutional layers are intended to learn and extract various features from the input image. The YOLO architecture's first few convolutional layers are responsible for extracting low-level features like edges and corners, while the higher convolutional layers are responsible for extracting high-level features like object parts and object shapes.

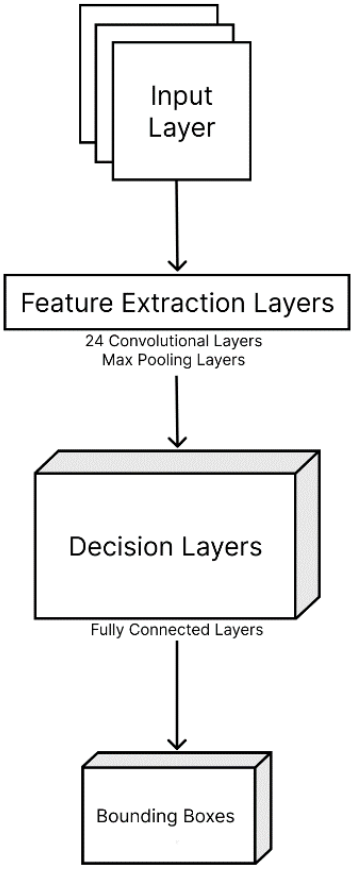


Fig 5.5: Architecture of YOLO Algorithm

The outputs of this layer are then sent to the decision layer, which makes forecasts for each grid cell in the final feature map. The decision layer in YOLO consists of a convolutional layer with a 1x1 kernel size that is applied to the output of the feature

extraction layers. This layer produces a grid of values that represent the predicted bounding boxes for objects in the image. Each grid cell in the output corresponds to a region of the input image, and each grid cell predicts a fixed number of bounding boxes along with their associated class probabilities.

For each predicted bounding box, the decision layer outputs the coordinates of the box, its width and height, and the confidence score for the object being present in that box. The confidence score represents the probability that the predicted bounding box contains an object. The decision layer also predicts the class probabilities for each bounding box. For each bounding box, the model predicts a probability distribution over all possible object classes. The class with the highest probability is then assigned to the object contained in the bounding box.

This enables the model to recognize and locate objects in the input picture. The actual choice regarding the bounding boxes and the discovered items is made by the completely connected layers.

The YOLO (You Only Look Once) algorithm is well-known for its fast processing times and great accuracy. The newest version of the algorithm, YOLOv5, is based on the PyTorch library and offers significant improvements over previous versions. To train YOLOv5 on our customised dataset, we must first obtain it in Yolov5 pytorch format (Fig 5.2.1.2) and then use PyTorch to fine-tune the pre-trained model.

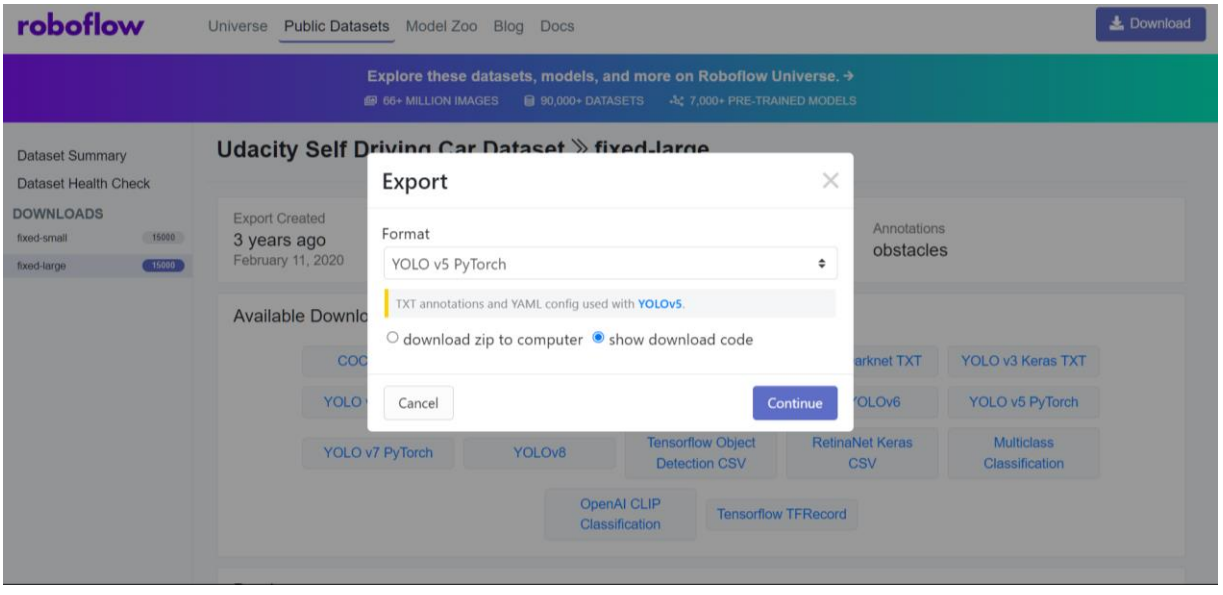


Fig 5.6: Downloading the Dataset in YOLOv5 PyTorch Format

The YOLO (You Only Look Once) algorithm is well-known for its fast processing times and great accuracy. The newest version of the algorithm, YOLOv5, is based on the PyTorch library and offers significant improvements over previous versions. To train YOLOv5 on our customised dataset, we must first obtain it in Yolov5 pytorch format (Fig 5.2.2) and then use PyTorch to fine-tune the pre-trained model.

Training a customized dataset using the YOLOv5 algorithm (Fig 5.2.1.3) involves providing information to the train.py file about various aspects of the training process. This includes specifying the location of the dataset, defining the model architecture, setting hyperparameters, and configuring the training schedule.

Fig 5.7: Training the YOLO Model

After defining the model architecture and configuring the training settings, the YOLOv5 model will be trained. The model will learn to recognize items in your photographs during training and will change its weights to minimize training loss.

Once trained, the YOLO model is used to identify numerous objects in real-time by feeding it testing images.

5.2.2 Training in Faster R-CNN Algorithm

The newer version of CNN is the Faster R-CNN. The Faster R-CNN processes the input image to produce a vector representation. With the use of picture annotation, the R-CNN network is trained on a datasets for the focusing component of images.

Depending on the diversity of each image, we change the value of the number of image areas.

After the fast R-convolution CNN's layer, there is a region proposal network (Fig 5.2.2.1) in which a portion of the input image's feature pixel matrix is referred to as a proposal. The RPN then generates a set of object proposals, which are regions in the feature map that are likely to contain objects. These proposals are refined by the RCNN, which performs classification and regression to predict the class and location of the objects within the proposals. These proposals are then passed on to the Region of Interest (ROI) pooling layer.

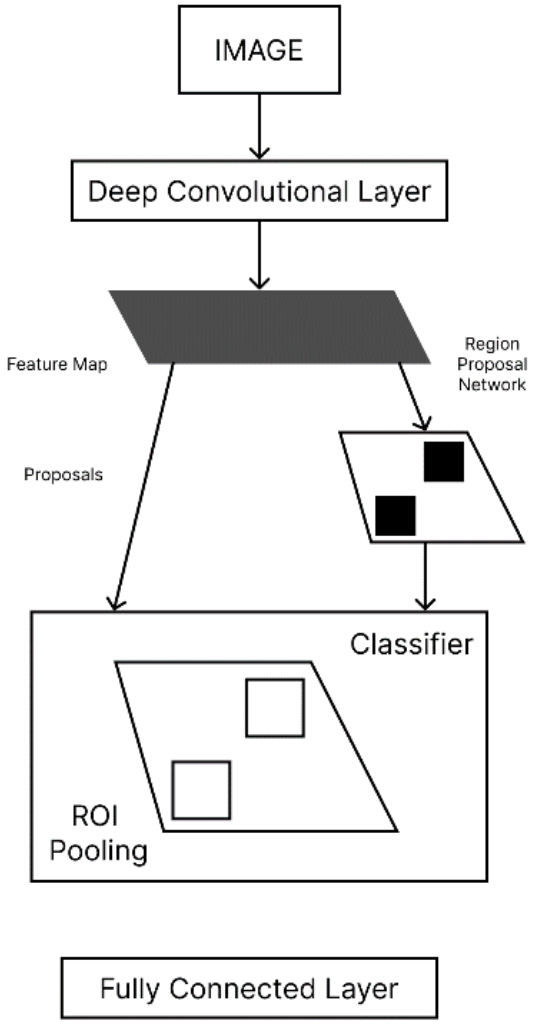


Fig 5.8: Architecture of Faster R-CNN Algorithm

The ROI pooling layer is responsible for extracting fixed-sized features from the proposals generated by the RPN. The layer divides each proposal into a fixed number of sub-windows or regions and applies max pooling on each sub-window to

generate a fixed-size feature map. This ensures that the features extracted from each proposal have a fixed size and are compatible with the fully connected layers in the network.

The output of the ROI pooling layer is a set of fixed-size feature maps for each proposal, which are then passed to a set of fully connected layers for classification and bounding box regression. These layers process the features to predict the class and location of objects in the input image.

The custom Faster R-CNN Model is trained using a Detectron 2 Facebook AI model. Detectron 2 is a flexible and modular platform for creating and testing new object detection and segmentation models that is built on top of PyTorch, a popular deep learning framework. Detectron 2 comes with a large number of pre-trained models as well as tools for training and assessing custom models. It enables a wide range of cutting-edge algorithms, including Faster R-CNN, Mask R-CNN, and RetinaNet. To train Faster R-CNN on our customised dataset, we must first obtain it in COCO JSON format (Fig 5.2.2.2) and then use detectron 2 to train the model.

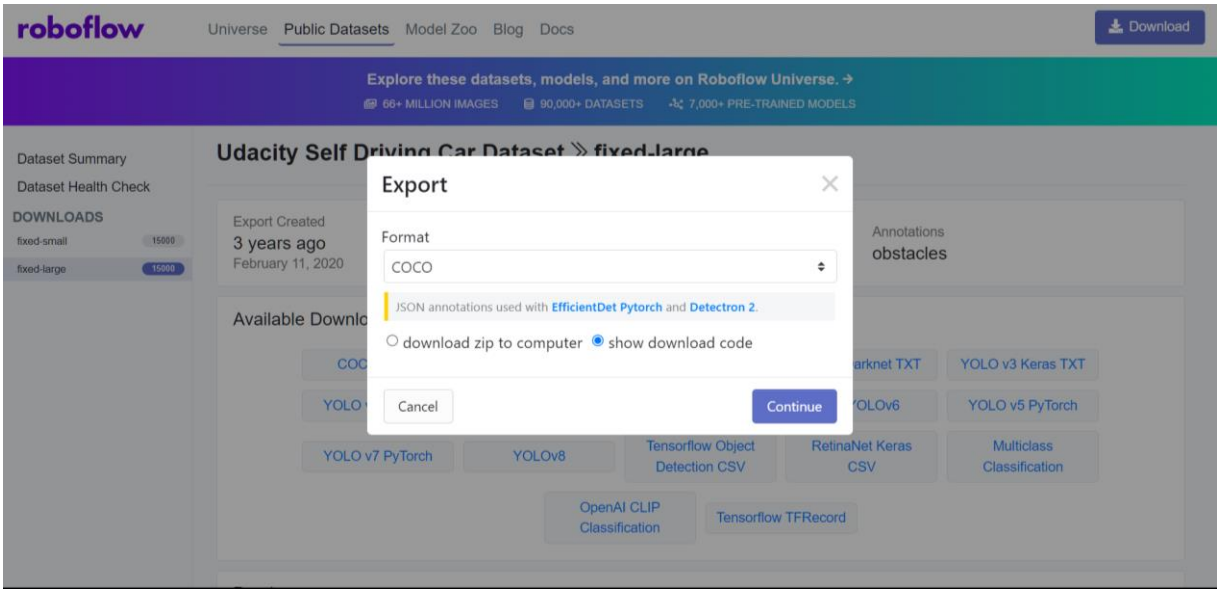


Fig 5.9: Downloading the Dataset in COCO JSON Format

When training a Faster R-CNN model for object detection on a dataset (Fig 5.2.2.3) , there are several configurations that need to be set, including the architecture of the model, hyperparameters, and training settings.

One important configuration is the number of training iterations, which determines the number of times the model will be trained on the dataset. The number of training

iterations is set based on factors such as the size and complexity of the dataset, the available computational resources, and the desired level of accuracy. This involves specifying the layers and connections between them that make up the model, as well as setting any additional hyperparameters specific to the Faster R-CNN architecture.

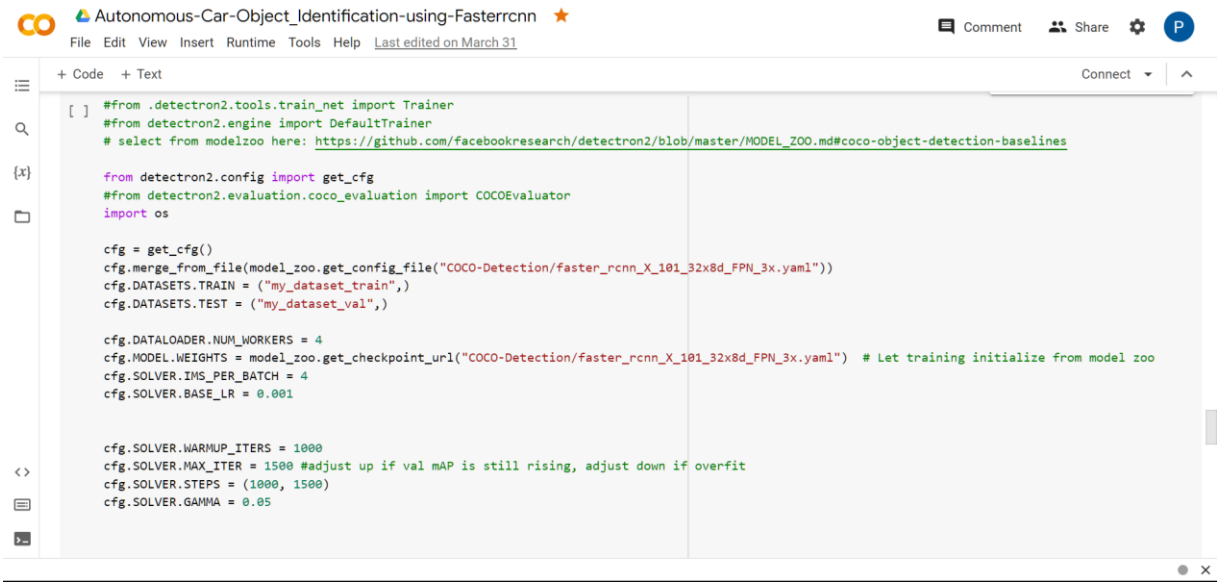


Fig 5.10: Training the Faster R-CNN Model

After defining the model architecture and configuring the training settings, the Faster R-CNN model will be trained. The model will learn to recognize items in your photographs during training and will change its weights to minimize training loss.

Once trained, the Faster R-CNN model is used to identify numerous objects in real-time by feeding it testing images.

5.3 Testing

5.3.1 Testing YOLO Model

Testing all the images from a testing dataset using the function obtained from training the data by YOLO Model (Fig 5.3.1.1) is an essential step in evaluating the performance of the object detection algorithm. This process typically involves running the trained YOLO model on each image in the testing dataset and generating predictions for the objects detected in each image.

The output of the testing process will typically include the predicted bounding boxes around each detected object, as well as the corresponding class labels and

confidence scores. This information can be used to evaluate the accuracy of the YOLO model in detecting objects in the testing dataset.

Fig 5.11: Testing the YOLO Model

The results of the testing process are then stored in a results folder (Fig 5.3.1.2), where they can be easily accessed and downloaded.

Fig 5.12: Storing the results of YOLO Model

5.3.2 Testing Faster R-CNN Model

The COCO (Common Objects in Context) validation evaluation is a common benchmark for evaluating the performance of object detection models. Importing a custom Trainer module can be useful if need to modify the testing process or implement a custom evaluation metric.

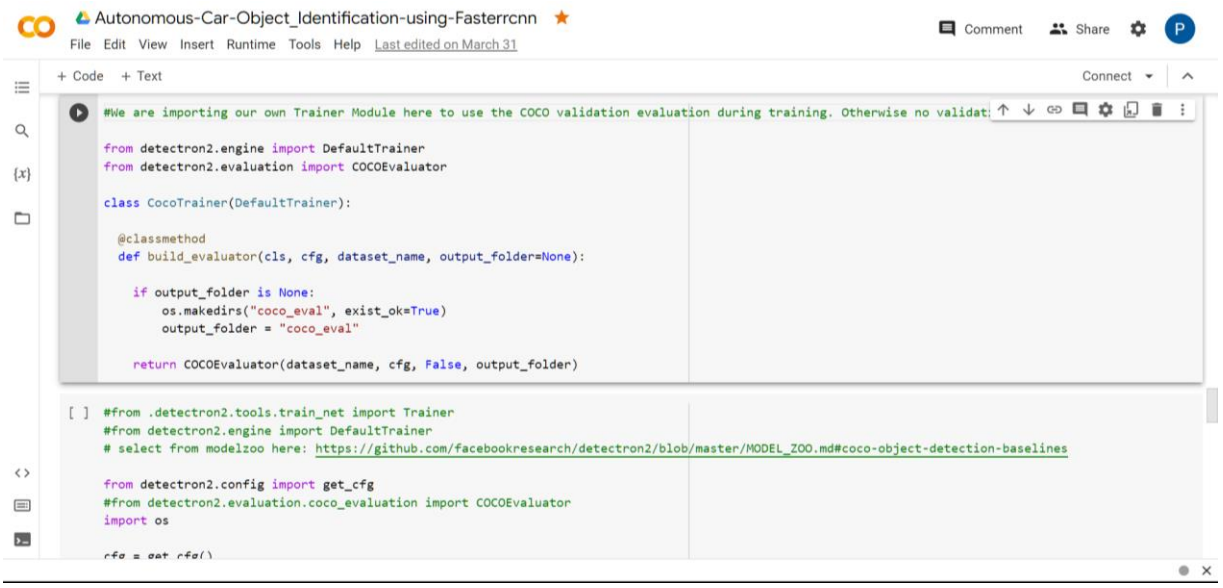


Fig 5.13: Testing and Storing the results of Faster R-CNN Model

Creating a directory called "coco_eval" (Fig 5.3.2.1) to store the outputs of COCO validation is a good practice for organizing and managing the evaluation results. The outputs of COCO validation typically include metrics such as mean average precision (mAP), which can be used to evaluate the performance of the object detection model. Storing these outputs in a separate directory can make it easier to track and compare the performance of the model over time.

CHAPTER 6

RESULTS AND DISCUSSION

6.1 EVALUATION OF THE PROPOSED MODEL

TensorBoard is a web-based visualization tool that comes with the TensorFlow machine learning library. It allows to monitor the progress of our YOLO and Faster R-CNN model in real-time by displaying the training losses and performance metrics in graphical form. It is also used to analyze our model's performance and identify potential areas for improvement.

During training, the training losses and performance metrics are saved to TensorBoard by including TensorBoard callbacks in our code (Fig 6.1,6.2). These callbacks will write the data to a log file, which TensorBoard can then read and display as graphs and other visualizations.

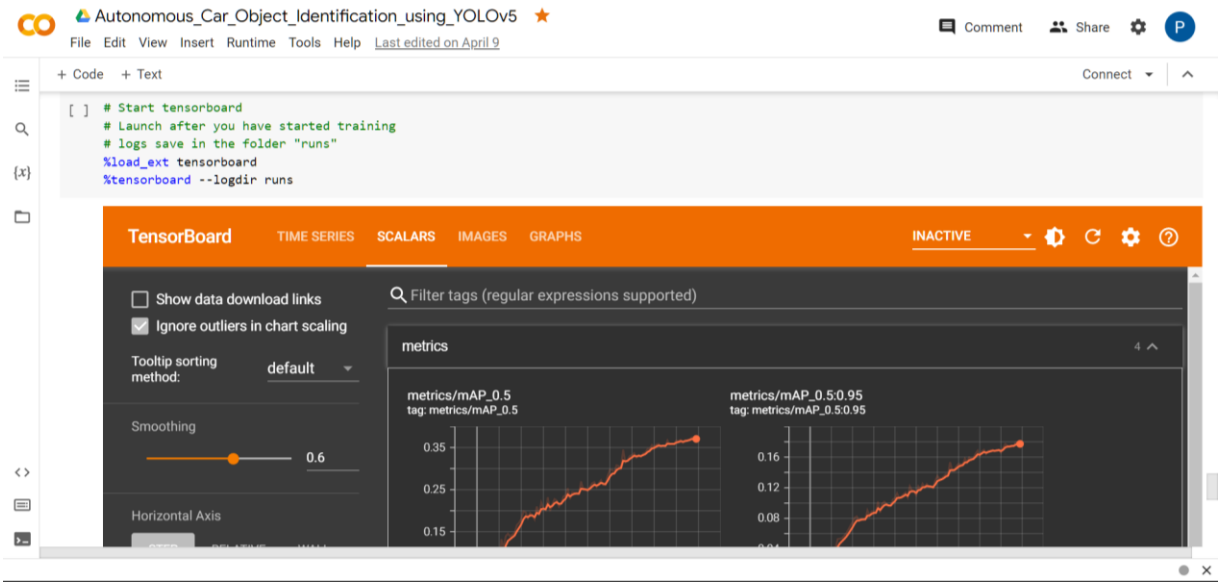


Fig 6.1: Launch TensorBoard in YOLO Model

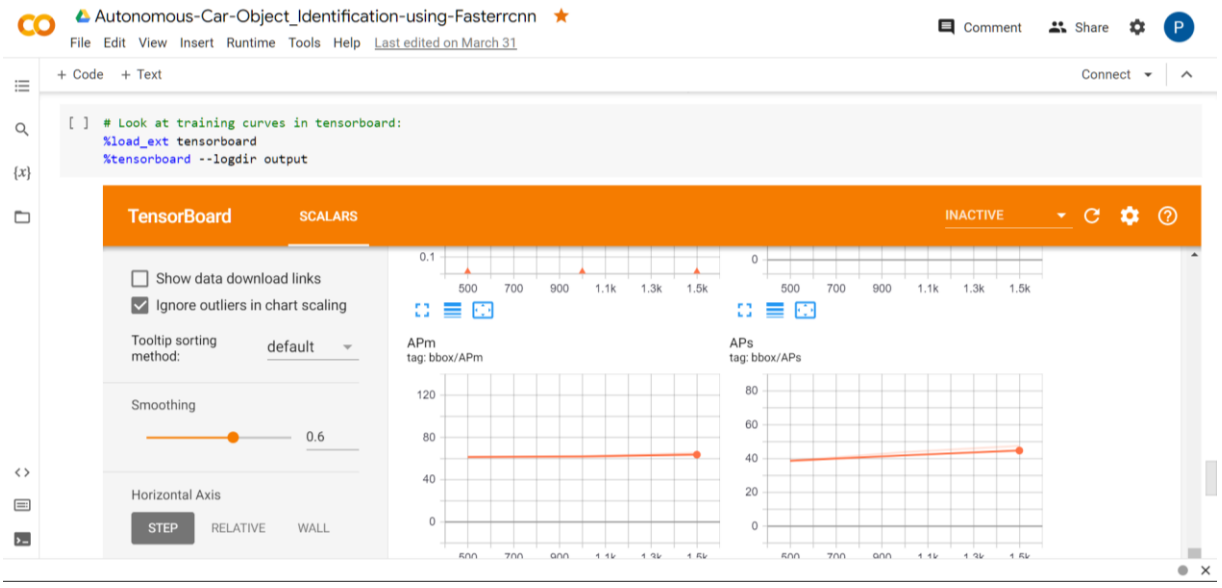


Fig 6.2: Launch TensorBoard in* Faster R-*CNN Model

Once the training data are saved to TensorBoard, we can use the tool to monitor and analyze the performance of YOLO and Faster R-CNN model. We can view the training losses and performance metrics over time, compare different models, and explore the relationships between different variables. This information can help us optimize the YOLO and Faster R-CNN model and improve its accuracy and performance.

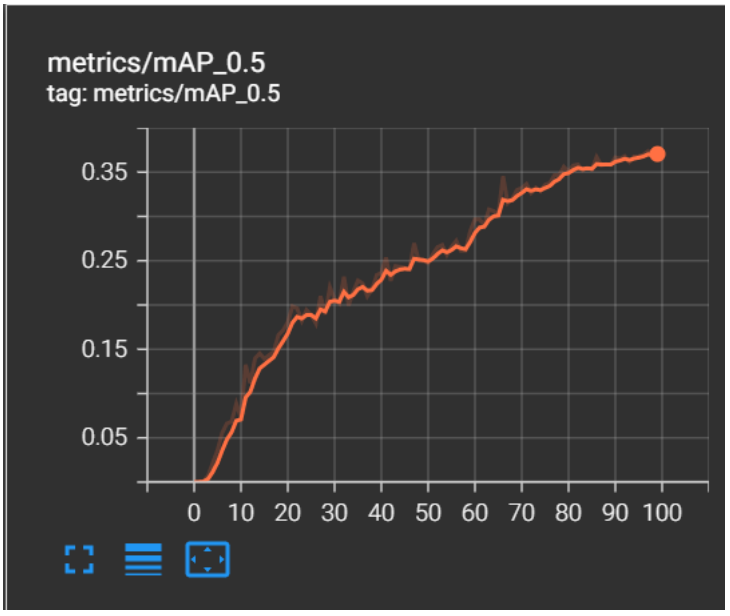


Fig 6.3: Performance metrics of YOLO Model

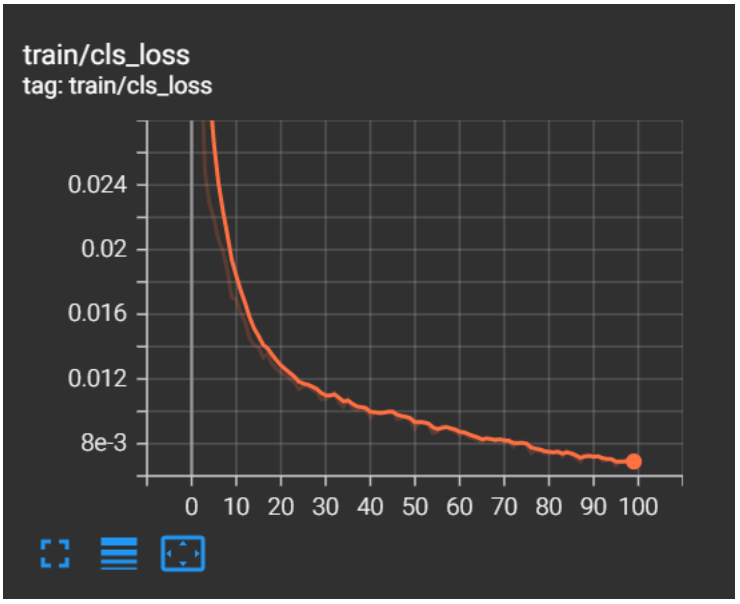


Fig 6.4: Training Loss of YOLO Model

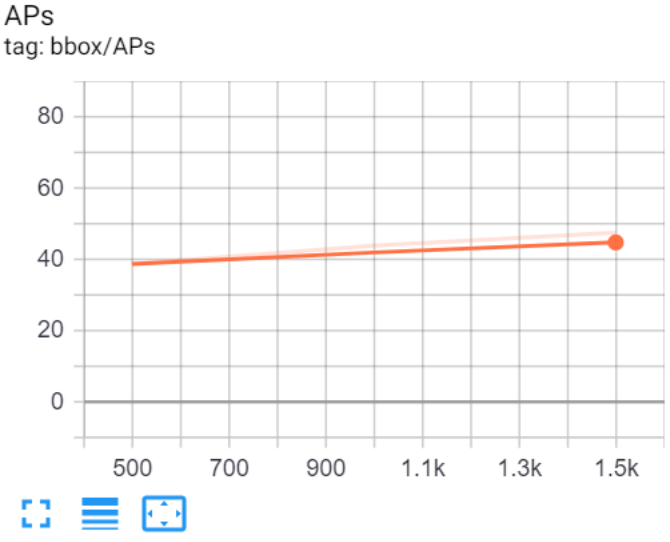


Fig 6.5: Performance metrics of* Faster R-*CNN Model

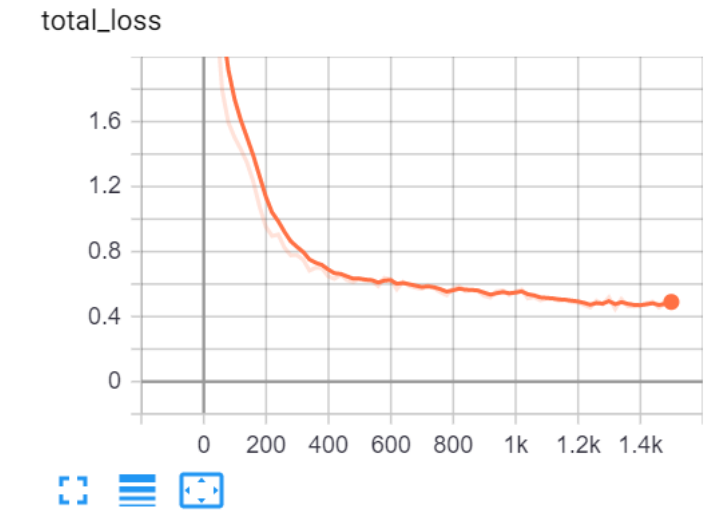


Fig 6.6: Training Loss of* Faster R-*CNN Model

The TensorBoard evaluation results shows the performance metrics are constantly increasing and the total loss of training is very low, it means that our both YOLO and Faster R-CNN models are improving over time and is learning the patterns in the data very well. This is generally a good indication that our models are performing very well on the training data.

The evaluation results showed that Faster R-CNN has greater training accuracy and lower training loss than YOLO, and that Faster R-CNN has detected more classes quicker than YOLO due to its simple architecture, then these are positive indications that CNN may be a better model for the detecting / identifying objects around the autonomous car and datasets being used.

The output of the object detected images is saved in a folder called "results," and it is simple to download them. Some of the output images are shown below.

Fig 6.7: Output Image 1

Fig 6.8: Output Image 2

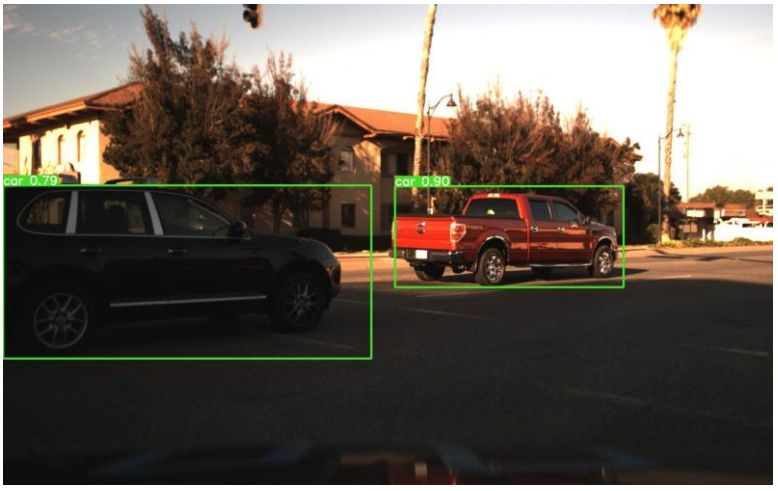


Fig 6.9: Output Image 3

CHAPTER 7

CONCLUSION AND FUTURE WORK

7.1 CONCLUSION

In Conclusion , we have successfully built and trained custom YOLO and Faster R-CNN models using datasets related to autonomous driving. We have also used TensorBoard to evaluate the models' performance during training and validation, and the evaluation results have shown that Faster R-CNN has greater training accuracy and lower training loss than YOLO. Additionally, Faster R-CNN has detected more classes quicker than YOLO due to its simpler architecture.

These are positive indications that Faster R-CNN may be a better model for the specific task and dataset being used. However, it is important to keep in mind that other factors such as computational efficiency, model interpretability, and ethical considerations may also need to be taken into account when selecting a final model.

7.2 FUTURE WORK

Detecting objects around an autonomous car using YOLO and CNN is an important step towards building a fully autonomous driving system. Here are some potential areas for future work in this domain:

1. **Object tracking:** After detecting objects around the car, it is important to track their movements over time. Object tracking can be used to predict the future location of objects and help the car avoid collisions. Future work could focus on developing robust object tracking algorithms that can handle occlusions and other challenges.

2. **Semantic segmentation:** In addition to object detection, semantic segmentation can provide more detailed information about the environment around the car, such as the layout of the road and other structures. Future work could focus on developing real-time semantic segmentation algorithms that can run on the

limited computing resources available in an autonomous car.

3. **Multi-sensor fusion:** Autonomous cars rely on multiple sensors, such as cameras, lidar, and radar, to perceive the environment. Multi-sensor fusion can combine information from multiple sensors to provide a more comprehensive understanding of the environment. Future work could focus on developing algorithms for sensor fusion that can handle the noisy and incomplete data from the sensors.

4. **Reinforcement learning:** Reinforcement learning can be used to train the car to make decisions based on the sensor data it receives. Future work could focus on developing reinforcement learning algorithms that can handle the high-dimensional input from the sensors and can learn to make safe and efficient driving decisions.

5. **Robustness and safety:** Autonomous cars need to operate safely in a wide range of environments and conditions, such as different weather conditions, road layouts, and traffic situations. Future work could focus on developing robust and safe autonomous driving systems that can handle unexpected events and edge cases.

These are just a few examples of the potential areas for future work after detecting objects around an autonomous car using YOLO and CNN. The specific directions will depend on the application and goals of the project.

REFERENCES

[1] S. Swetha and P. Sivakumar, "SSLA Based Traffic Sign and Lane Detection for Autonomous cars," 2021 International Conference on Artificial Intelligence and Smart Systems (ICAIS), Coimbatore, India, 2021, pp. 766-771, doi: 10.1109/ICAIS50930.2021.9396046.

[2] Fan, J. Lee, I. Jung and Y. Lee, "Improvement of Object Detection Based on Faster R-CNN and YOLO," 2021 36th International Technical Conference on Circuits/Systems, Computers and Communications (ITC-CSCC), Jeju, Korea (South), 2021, pp. 1-4, doi: 10.1109/ITC-CSCC52171.2021.9501480.

[3] B. Balakrishnan, R. Chelliah, M. Venkatesan and C. Sah, "Comparative Study On Various Architectures Of Yolo Models Used In Object Recognition," 2022 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS), Greater Noida, India, 2022, pp. 685-690, doi: 10.1109/ICCCIS56430.2022.10037635.

[4] K. R and N. S, "Pothole and Object Detection for an Autonomous Vehicle Using YOLO," 2021 5th International Conference on Intelligent Computing and Control Systems (ICICCS), Madurai, India, 2021, pp. 1585-1589, doi: 10.1109/ICICCS51141.2021.9432186.

[5] N. H. Sarhan and A. Y. Al-Omary, "Traffic light Detection using OpenCV and YOLO," 2022 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT), Sakheer, Bahrain, 2022, pp. 604-608, doi: 10.1109/3ICT56508.2022.9990723.

[6] M. Mostafa and M. Ghantous, "A YOLO Based Approach for Traffic Light Recognition for ADAS Systems," 2022 2nd International Mobile, Intelligent, and Ubiquitous Computing Conference (MIUCC), Cairo, Egypt, 2022, pp. 225-229, doi: 10.1109/MIUCC55081.2022.9781682.

[7] A. D. The and M. Yoo, "MissVoxelNet: 3D Object Detection for Autonomous Vehicle in Snow Conditions," 2022 Thirteenth International Conference on Ubiquitous and Future Networks (ICUFN), Barcelona, Spain, 2022, pp. 479-482, doi:

10.1109/ICUFN55119.2022.9829647.

[8] A. Agafonov and A. Yumaganov, "3D Objects Detection in an Autonomous Car Driving Problem," 2020 International Conference on Information Technology and Nanotechnology (ITNT), Samara, Russia, 2020, pp. 1-5, doi: 10.1109/ITNT49337.2020.9253253.

[9] G. Akyol, A. Kantarcı, A. E. Çelik and A. Cihan Ak, "Deep Learning Based, Real-Time Object Detection for Autonomous Driving," 2020 28th Signal Processing and Communications Applications Conference (SIU), Gaziantep, Turkey, 2020, pp. 1-4, doi: 10.1109/SIU49456.2020.9302500.

[10] M. Mobahi and S. H. Sadati, "An Improved Deep Learning Solution for Object Detection in Self-Driving Cars," 2020 28th Iranian Conference on Electrical Engineering (ICEE), Tabriz, Iran, 2020, pp. 1-5, doi: 10.1109/ICEE50131.2020.9260870.

[11] P. Cai, H. Wang, H. Huang, Y. Liu and M. Liu, "Vision-Based Autonomous Car Racing Using Deep Imitative Reinforcement Learning," in IEEE Robotics and Automation Letters, vol. 6, no. 4, pp. 7262-7269, Oct. 2021, doi: 10.1109/LRA.2021.3097345.

[12] I. Sonata, Y. Heryadi, W. Budiharto and A. Wibowo, "Street View Object Detection for Autonomous Car Steering Angle Prediction Using Convolutional Neural Network," 2021 1st International Conference on Computer Science and Artificial Intelligence (ICCSAI), Jakarta, Indonesia, 2021, pp. 367-372, doi: 10.1109/ICCSAI53272.2021.9609736.

[13] C. Zhao, S. Jiang, Y. Lei and C. Wang, "A Study on an Anthropomorphic Car-Following Strategy Framework of the Autonomous Coach in Mixed Traffic Flow," in IEEE Access, vol. 8, pp. 64653-64665, 2020, doi: 10.1109/ACCESS.2020.2985749.

[14] M. Masmoudi, H. Friji, H. Ghazzai and Y. Massoud, "A Reinforcement Learning Framework for Video Frame-Based Autonomous Car-Following," in IEEE Open Journal of Intelligent Transportation Systems, vol. 2, pp. 111-127, 2021, doi: 10.1109/OJITS.2021.3083201.

[15] R. Vieira, E. Argento and T. Revoredo, "Trajectory Planning For Car-like Robots Through Curve Parametrization And Genetic Algorithm Optimization With Applications To Autonomous Parking," in IEEE Latin America Transactions, vol. 20, no. 2, pp. 309-316, Feb. 2022, doi: 10.1109/TLA.2022.9661471.

[16] V. Androulakis, J. Sottile, S. Schafrik and Z. Agioutantis, "Concepts for Development of Autonomous Coal Mine Shuttle Cars," in IEEE Transactions on Industry Applications, vol. 56, no. 3, pp. 3272-3280, May-June 2020, doi: 10.1109/TIA.2020.2972786.

[17] S. Malik, H. A. Khattak, Z. Ameer, U. Shoaib, H. T. Rauf and H. Song, "Proactive Scheduling and Resource Management for Connected Autonomous Vehicles: A Data Science Perspective," in IEEE Sensors Journal, vol. 21, no. 22, pp. 25151-25160, 15 Nov.15, 2021, doi: 10.1109/JSEN.2021.3074785.

[18] S. Mane and S. Mangale, "Moving Object Detection and Tracking Using Convolutional Neural Networks," 2018 Second International Conference on Intelligent Computing and Control Systems (ICICCS), Madurai, India, 2018, pp. 1809-1813, doi: 10.1109/ICCONS.2018.8662921.

APPENDIX

SOURCE CODE:

1. YOLO Model

```
!git clone https://github.com/ultralytics/yolov5
!pip install -qr yolov5/requirements.txt
%cd yolov5
!pip install IPython

import torch
from IPython.display import Image, clear_output

clear_output()

%cd /content
!curl -
L "https://public.roboflow.com/ds/9Yz9MgkaMT?key=YNo79C6mBa" > roboflow.zip; unzip roboflow.zip; rm roboflow.zip

!mkdir train/
!mkdir train/images
!mkdir train/labels
!mkdir valid/
!mkdir valid/images
!mkdir valid/labels
!mkdir test/
!mkdir test/images
!mkdir test/labels

import os
import shutil

# allImages = glob.glob("/export/images/*.jpg")
allImages = []
for filename in os.listdir("export/images/"):
    if filename.endswith(".jpg"):
      allImages.append(filename)
trainImages = []
valImages = []
trainLabels = []
valLabels = []
testImages = []
testLabels = []
print(len(allImages))
testCount = int(0.2 * len(allImages)/10)
trainCount = len(allImages)/10 - testCount-1
valCount = int(0.2 * trainCount)
trainCount = trainCount - valCount-1

print(trainCount, valCount, testCount)

# create training Dataset
j = 0
k = 0
for i in range(0,int(len(allImages)/10)):
  if i < trainCount: # create training images
    trainImages.append(allImages[i])
  elif i >= trainCount and i<(valCount + trainCount):
    valImages.append(allImages[i])
    # j += 1
  else:
    testImages.append(allImages[i])
    # k += 1

print(len(trainImages), len(valImages), len(testImages))

# moving training images
```

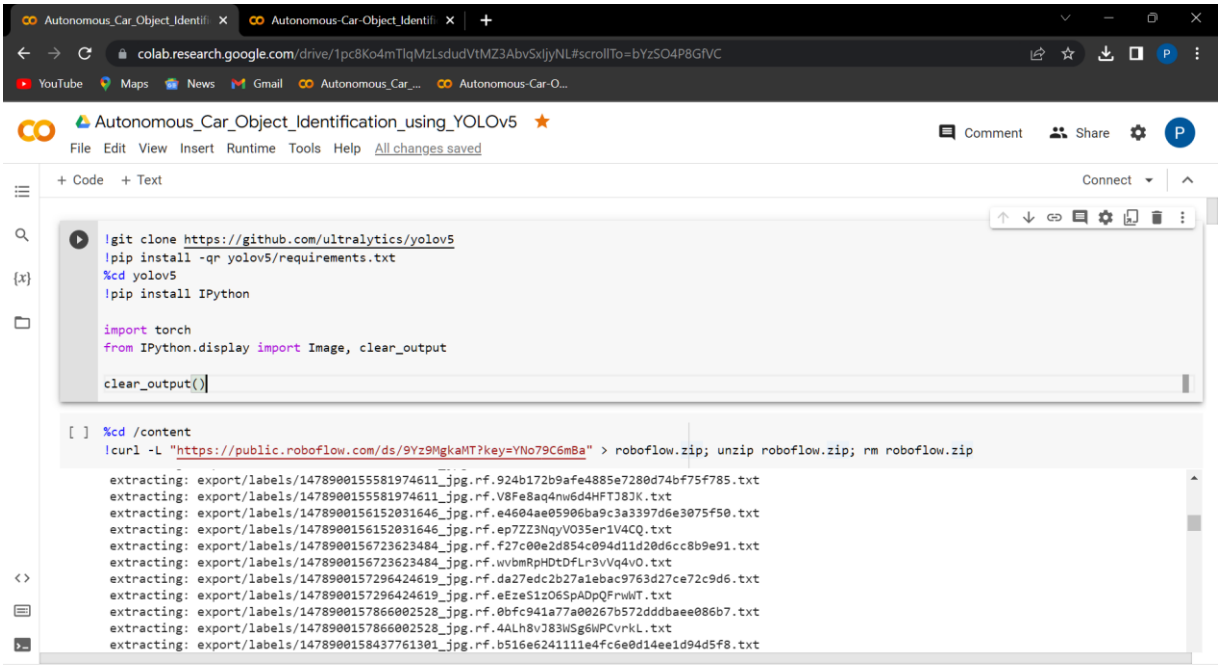
```
source = "export/images/"
destination = "train/images/"
for fileT in trainImages:
  shutil.move(source+fileT , destination+fileT)

source = "export/images/"
destination = "valid/images/"
for fileT in valImages:
  shutil.move(source+fileT , destination+fileT)

source = "export/images/"
destination = "test/images/"
for fileT in testImages:
  shutil.move(source+fileT , destination+fileT)

# moving labels
source = "export/labels/"
destination = "train/labels/"
for fileT in trainImages:
  fileT = fileT.replace(".jpg",".txt")
  shutil.move(source+fileT , destination+fileT)

source = "export/labels/"
destination = "valid/labels/"
for fileT in valImages:
  fileT = fileT.replace(".jpg",".txt")
  shutil.move(source+fileT , destination+fileT)

source = "export/labels/"
destination = "test/labels/"
for fileT in testImages:
  fileT = fileT.replace(".jpg",".txt")
  shutil.move(source+fileT , destination+fileT)

#extracting information from the roboflow file
%cat data.yaml

# define number of classes based on data.yaml
import yaml
with open("data.yaml", 'r') as stream:
    num_classes = str(yaml.safe_load(stream)['nc'])

%cat /content/yolov5/models/yolov5s.yaml

#customize iPython writefile so we can write variables
from IPython.core.magic import register_line_cell_magic

@register_line_cell_magic
def writetemplate(line,cell):
    with open(line, 'w') as f:
        f.write(cell.format(**globals()))

%%writetemplate /content/yolov5/models/custom_yolov5s.yaml

# train yolov5s on custom data for 100 epochs
# time its performance
%%time
%cd /content/yolov5/
!python train.py --img 416 --batch 16 --epochs 100  --data '../data.yaml' --cfg ./models/custom_yolov5s.yaml --
weights '' --name yolov5s_results  --cache

!python detect.py --weights runs/train/yolov5s_results/weights/best.pt --img 416 --conf 0.4 --source ../test/images

import glob
from IPython.display import Image, display

for imageName in glob.glob('/content/yolov5/inference/output/*.jpg'):
    display(Image(filename=imageName))
    print("\n")

# Start tensorboard
# Launch after you have started training
# logs save in the folder "runs"
%load_ext tensorboard
%tensorboard --logdir runs
```

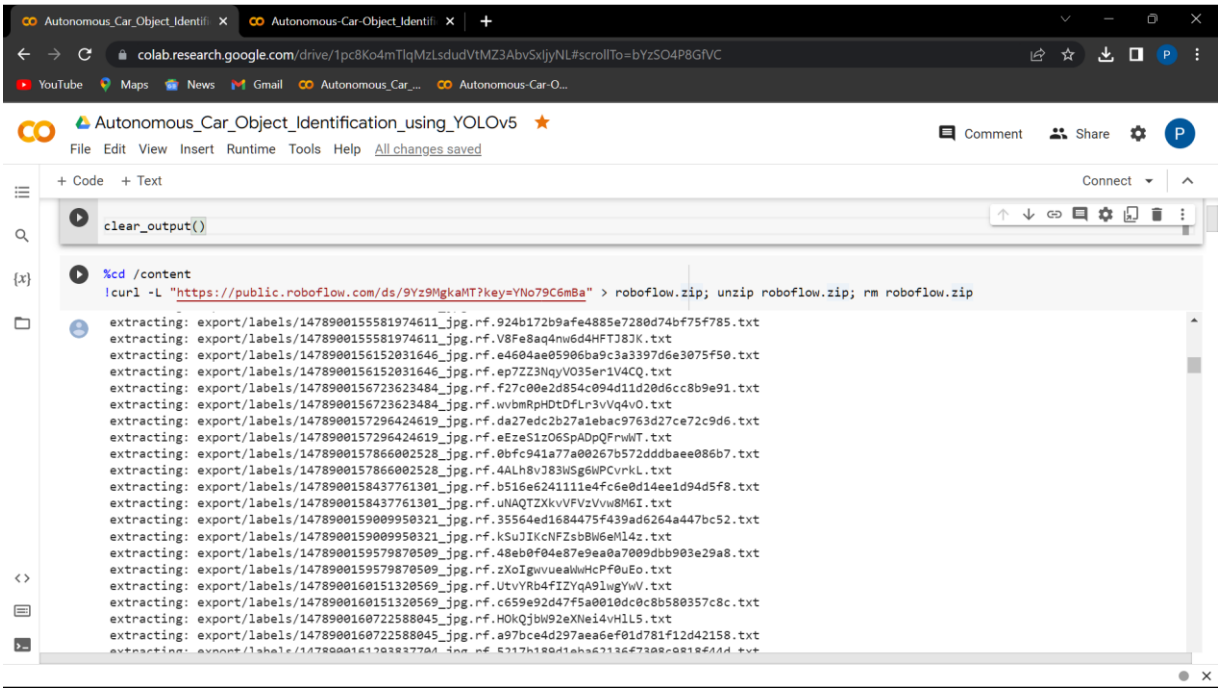
2. Faster R-CNN Model

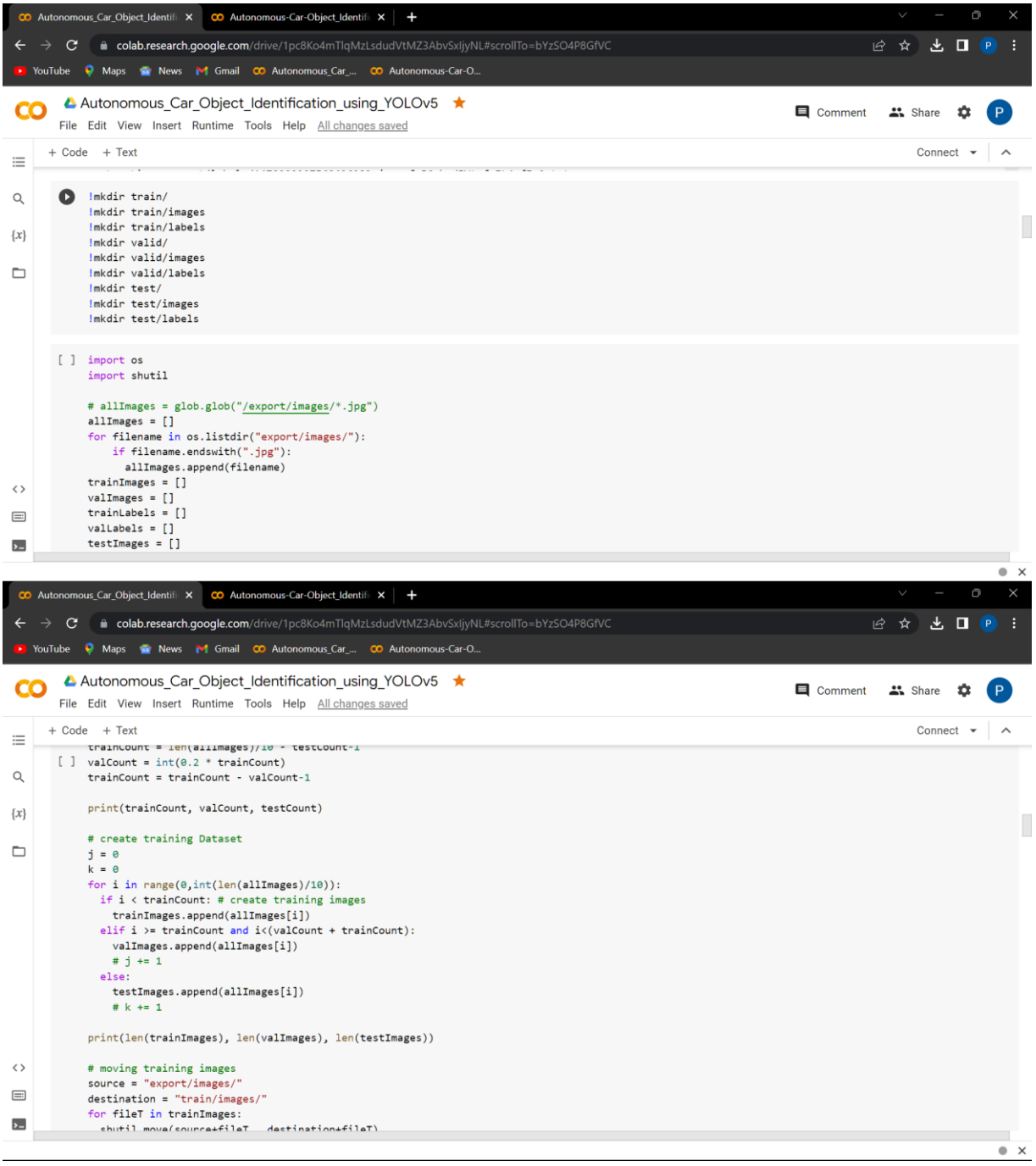
```
# install dependencies: (use cu101 because colab has CUDA 10.1)
!pip install -U torch==1.5 torchvision==0.6 -f https://download.pytorch.org/whl/cu101/torch_stable.html
!pip install cython pyyaml==5.1
!pip install -U 'git+https://github.com/cocodataset/cocoapi.git#subdirectory=PythonAPI'
import torch, torchvision
print(torch.__version__, torch.cuda.is_available())
!gcc --version
# opencv is pre-installed on colab
from IPython.display import clear_output
clear_output()

# install detectron2:
!pip install detectron2==0.1.3 -f https://dl.fbaipublicfiles.com/detectron2/wheels/cu101/torch1.5/index.html

# You may need to restart your runtime prior to this, to let your installation take effect
# Some basic setup:
# Setup detectron2 logger
import detectron2
from detectron2.utils.logger import setup_logger
setup_logger()

# import some common libraries
import numpy as np
import cv2
import random
from google.colab.patches import cv2_imshow

# import some common detectron2 utilities
from detectron2 import model_zoo
from detectron2.engine import DefaultPredictor
from detectron2.config import get_cfg
from detectron2.utils.visualizer import Visualizer
from detectron2.data import MetadataCatalog
from detectron2.data.catalog import DatasetCatalog

!curl -
L "https://public.roboflow.com/ds/Wd21F3AObk?key=EnSFtk9CJD" > roboflow.zip; unzip roboflow.zip; rm roboflow.zip

from detectron2.data.datasets import register_coco_instances
register_coco_instances("my_dataset_train", {}, "/content/export/_annotations.coco.json", "/content/export")
register_coco_instances("my_dataset_val", {}, "/content/export/_annotations.coco.json", "/content/export")
register_coco_instances("my_dataset_test", {}, "/content/export/_annotations.coco.json", "/content/export")

#visualize training data
my_dataset_train_metadata = MetadataCatalog.get("my_dataset_train")
dataset_dicts = DatasetCatalog.get("my_dataset_train")

import random
from detectron2.utils.visualizer import Visualizer

for d in random.sample(dataset_dicts, 3):
    img = cv2.imread(d["file_name"])
    visualizer = Visualizer(img[:, :, ::-1], metadata=my_dataset_train_metadata, scale=0.5)
    vis = visualizer.draw_dataset_dict(d)
    cv2_imshow(vis.get_image()[:, :, ::-1])

#We are importing our own Trainer Module here to use the COCO validation evaluation during training. Otherwise no vali
dation eval occurs.

from detectron2.engine import DefaultTrainer
from detectron2.evaluation import COCOEvaluator

class CocoTrainer(DefaultTrainer):

  @classmethod
  def build_evaluator(cls, cfg, dataset_name, output_folder=None):

    if output_folder is None:
        os.makedirs("coco_eval", exist_ok=True)
        output_folder = "coco_eval"

    return COCOEvaluator(dataset_name, cfg, False, output_folder)
```

```
#from .detectron2.tools.train_net import Trainer
#from detectron2.engine import DefaultTrainer
# select from modelzoo here: https://github.com/facebookresearch/detectron2/blob/master/MODEL_ZOO.md#coco-object-
detection-baselines

from detectron2.config import get_cfg
#from detectron2.evaluation.coco_evaluation import COCOEvaluator
import os

cfg = get_cfg()
cfg.merge_from_file(model_zoo.get_config_file("COCO-Detection/faster_rcnn_X_101_32x8d_FPN_3x.yaml"))
cfg.DATASETS.TRAIN = ("my_dataset_train",)
cfg.DATASETS.TEST = ("my_dataset_val",)

cfg.DATALOADER.NUM_WORKERS = 4
cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url("COCO-
Detection/faster_rcnn_X_101_32x8d_FPN_3x.yaml")  # Let training initialize from model zoo
cfg.SOLVER.IMS_PER_BATCH = 4
cfg.SOLVER.BASE_LR = 0.001

cfg.SOLVER.WARMUP_ITERS = 1000
cfg.SOLVER.MAX_ITER = 1500 #adjust up if val mAP is still rising, adjust down if overfit
cfg.SOLVER.STEPS = (1000, 1500)
cfg.SOLVER.GAMMA = 0.05



cfg.MODEL.ROI_HEADS.BATCH_SIZE_PER_IMAGE = 64
cfg.MODEL.ROI_HEADS.NUM_CLASSES = 11 #your number of classes + 1

cfg.TEST.EVAL_PERIOD = 500

os.makedirs(cfg.OUTPUT_DIR, exist_ok=True)
trainer = CocoTrainer(cfg)
trainer.resume_or_load(resume=False)
trainer.train()

# Look at training curves in tensorboard:
%load_ext tensorboard
%tensorboard --logdir output
```

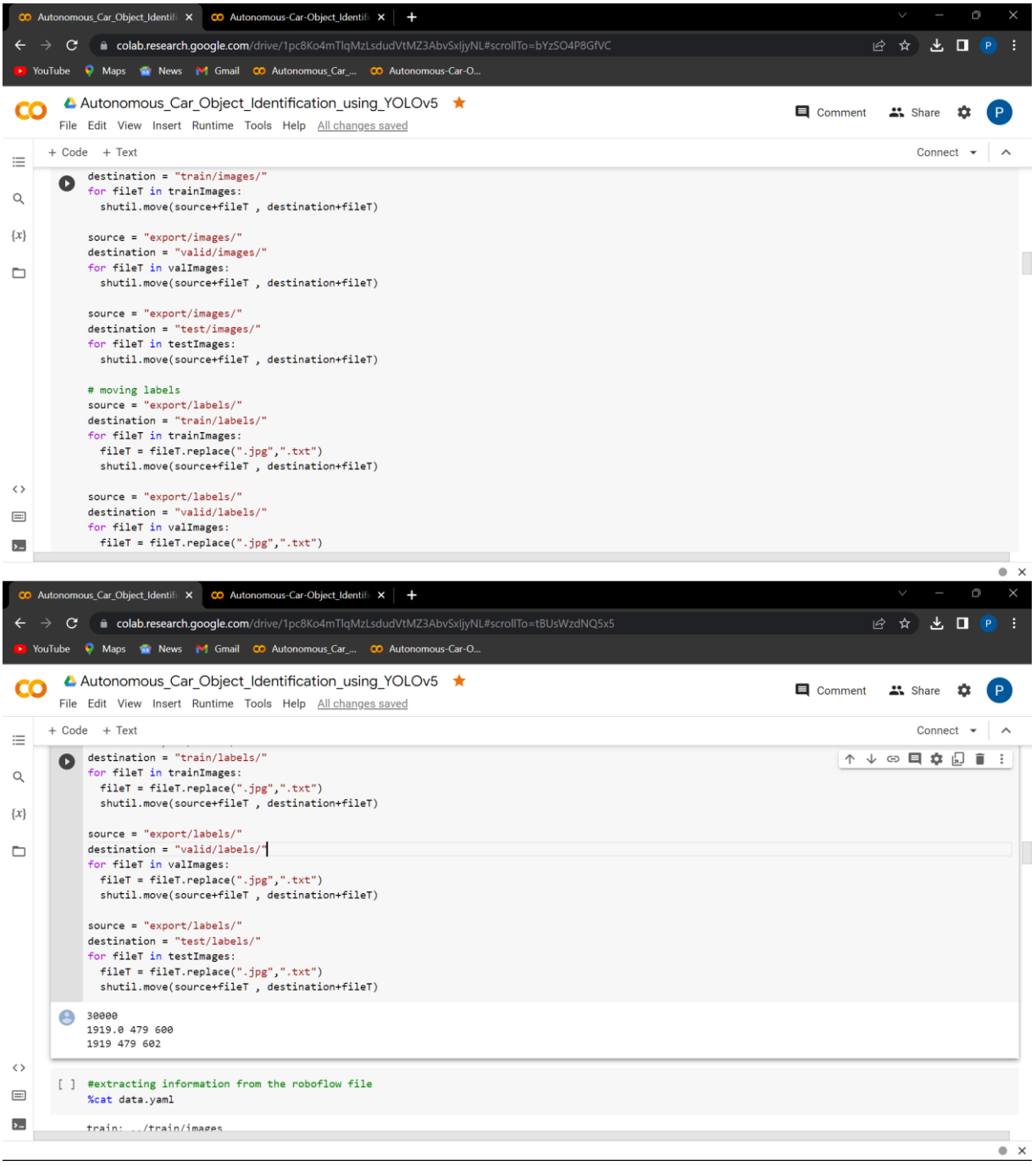
SCREENSHOTS:



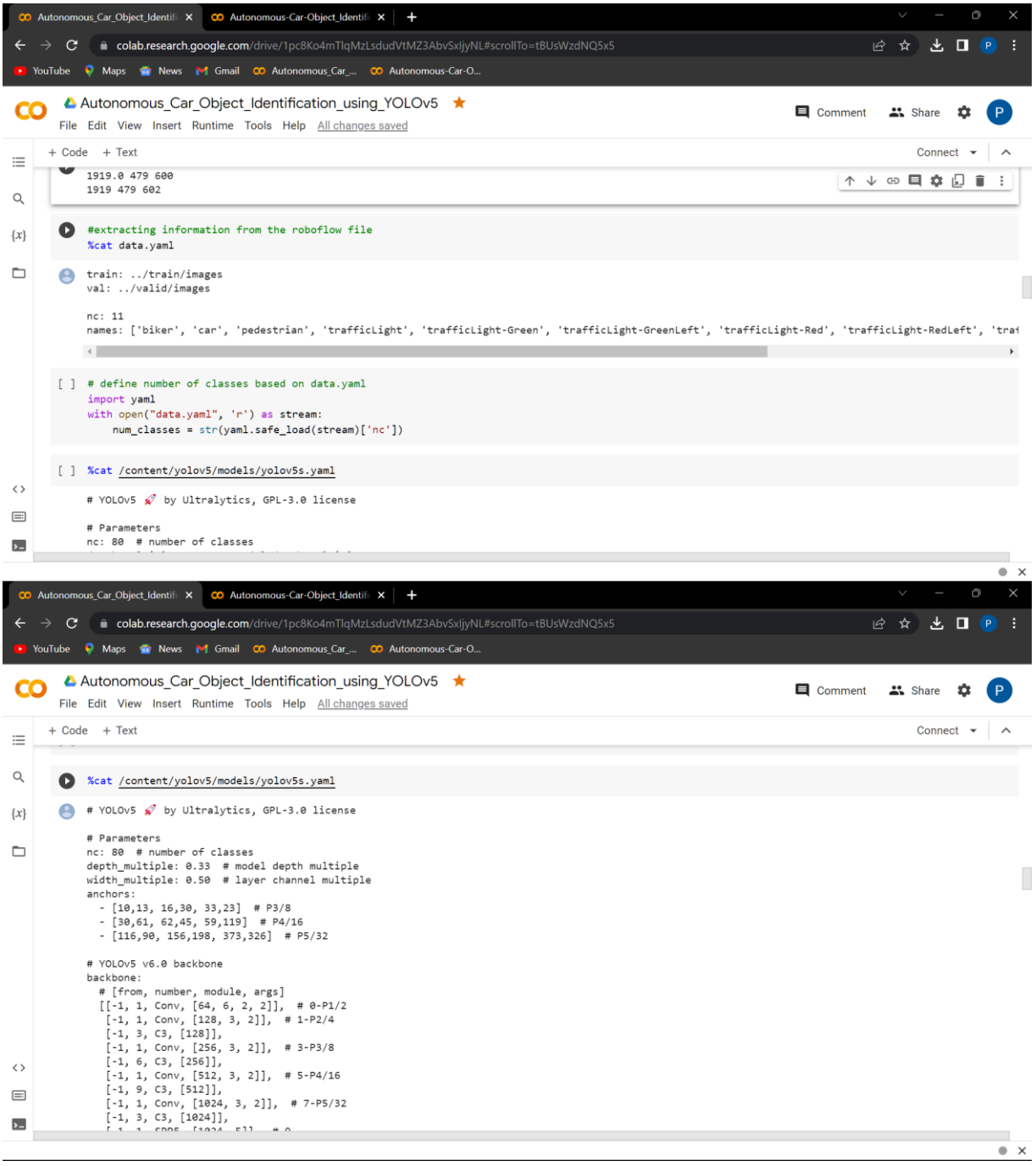



Autonomous_Car_Object_Identifi
Autonomous-Car-Object_Identifi
colab.research.google.com/drive/1pc8Ko4mTlqMzLsdudVtMZ3AbvSxljyNL#scrollTo=bYzSO4P8GfVC
YouTube
Maps
News
Gmail
Autonomous_Car_...
Autonomous-Car-O...
Autonomous_Car_Object_Identification_using_YOLOv5
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code
+ Text
Connect
!mkdir train/
!mkdir train/images
!mkdir train/labels
!mkdir valid/
!mkdir valid/images
!mkdir valid/labels
!mkdir test/
!mkdir test/images
!mkdir test/labels
[ ]
import os
import shutil
# allImages = glob.glob("/export/images/*.jpg")
allImages = []
for filename in os.listdir("export/images/"):
if filename.endswith(".jpg"):
allImages.append(filename)
trainImages = []
valImages = []
trainLabels = []
valLabels = []
testImages = []

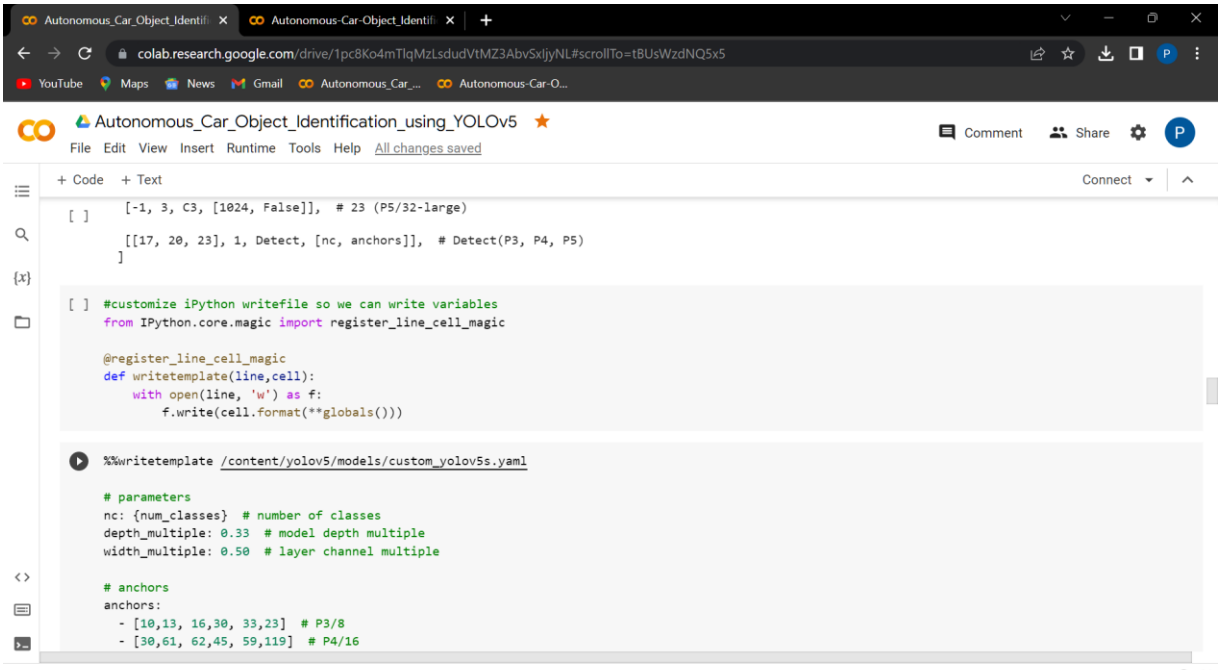
Autonomous_Car_Object_Identifi
Autonomous-Car-Object_Identifi
colab.research.google.com/drive/1pc8Ko4mTlqMzLsdudVtMZ3AbvSxljyNL#scrollTo=bYzSO4P8GfVC
YouTube
Maps
News
Gmail
Autonomous_Car_...
Autonomous-Car-O...
Autonomous_Car_Object_Identification_using_YOLOv5
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code
+ Text
Connect
[ ]
valCount = int(0.2 * trainCount)
trainCount = trainCount - valCount-1
print(trainCount, valCount, testCount)
# create training Dataset
j = 0
k = 0
for i in range(0,int(len(allImages)/10)):
if i < trainCount: # create training images
trainImages.append(allImages[i])
elif i >= trainCount and i<(valCount + trainCount):
valImages.append(allImages[i])
# j += 1
else:
testImages.append(allImages[i])
# k += 1
print(len(trainImages), len(valImages), len(testImages))
# moving training images
source = "export/images/"
destination = "train/images/"
for fileT in trainImages:

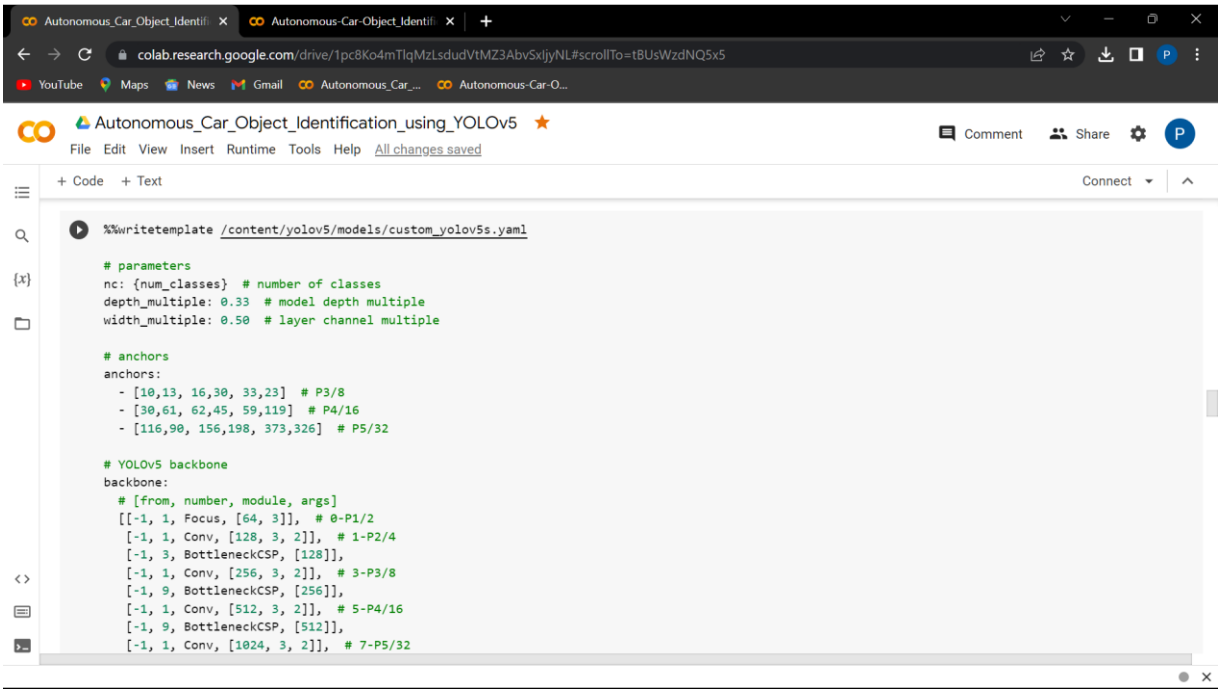

Autonomous_Car_Object_Identifi
Autonomous-Car-Object_Identifi
colab.research.google.com/drive/1pc8Ko4mTlqMzLsdudVtMZ3AbvSxljyNL#scrollTo=bYzSO4P8GfVC
YouTube
Maps
News
Gmail
Autonomous_Car_...
Autonomous-Car-O...
Autonomous_Car_Object_Identification_using_YOLOv5
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
Connect
destination = "train/images/"
for fileT in trainImages:
shutil.move(source+fileT , destination+fileT)
source = "export/images/"
destination = "valid/images/"
for fileT in valImages:
shutil.move(source+fileT , destination+fileT)
source = "export/images/"
destination = "test/images/"
for fileT in testImages:
shutil.move(source+fileT , destination+fileT)
# moving labels
source = "export/labels/"
destination = "train/labels/"
for fileT in trainImages:
fileT = fileT.replace(".jpg",".txt")
shutil.move(source+fileT , destination+fileT)
source = "export/labels/"
destination = "valid/labels/"
for fileT in valImages:
fileT = fileT.replace(".jpg",".txt")

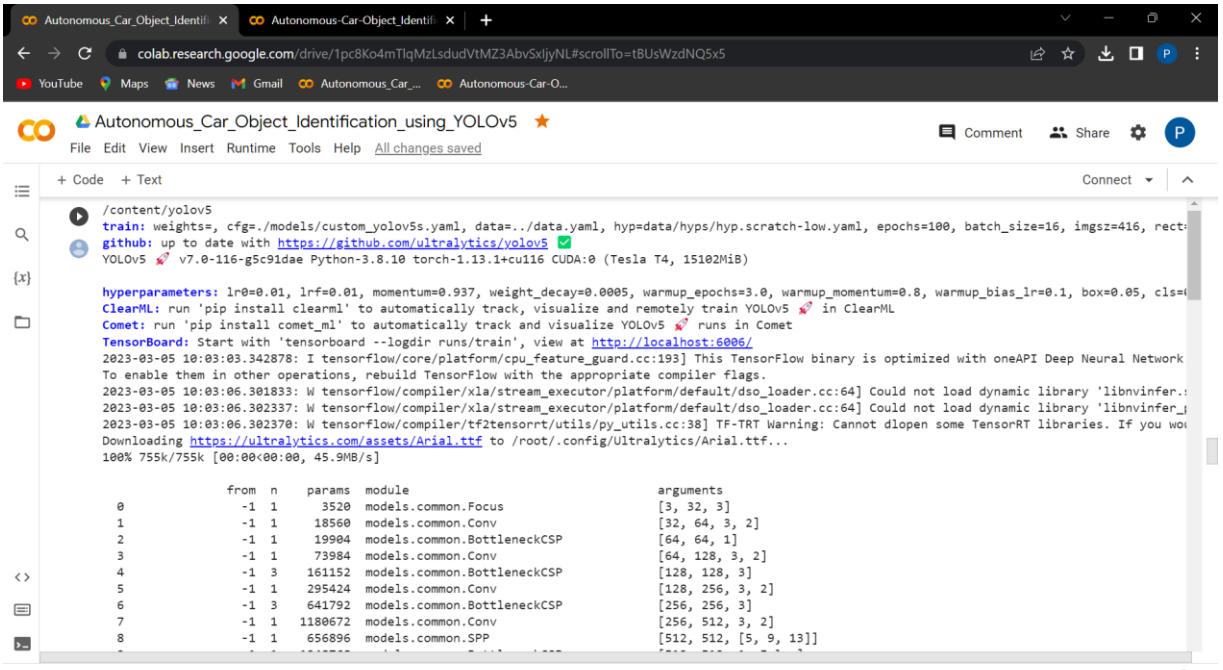
Autonomous_Car_Object_Identifi
Autonomous-Car-Object_Identifi
colab.research.google.com/drive/1pc8Ko4mTlqMzLsdudVtMZ3AbvSxljyNL#scrollTo=tBUsWzdNQ5x5
YouTube
Maps
News
Gmail
Autonomous_Car_...
Autonomous-Car-O...
Autonomous_Car_Object_Identification_using_YOLOv5
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
Connect
destination = "train/labels/"
for fileT in trainImages:
fileT = fileT.replace(".jpg",".txt")
shutil.move(source+fileT , destination+fileT)
source = "export/labels/"
destination = "valid/labels/"
for fileT in valImages:
fileT = fileT.replace(".jpg",".txt")
shutil.move(source+fileT , destination+fileT)
source = "export/labels/"
destination = "test/labels/"
for fileT in testImages:
fileT = fileT.replace(".jpg",".txt")
shutil.move(source+fileT , destination+fileT)
30000
1919.0 479 600
1919 479 602
[ ] #extracting information from the roboflow file
%cat data.yaml
train: ../train/images

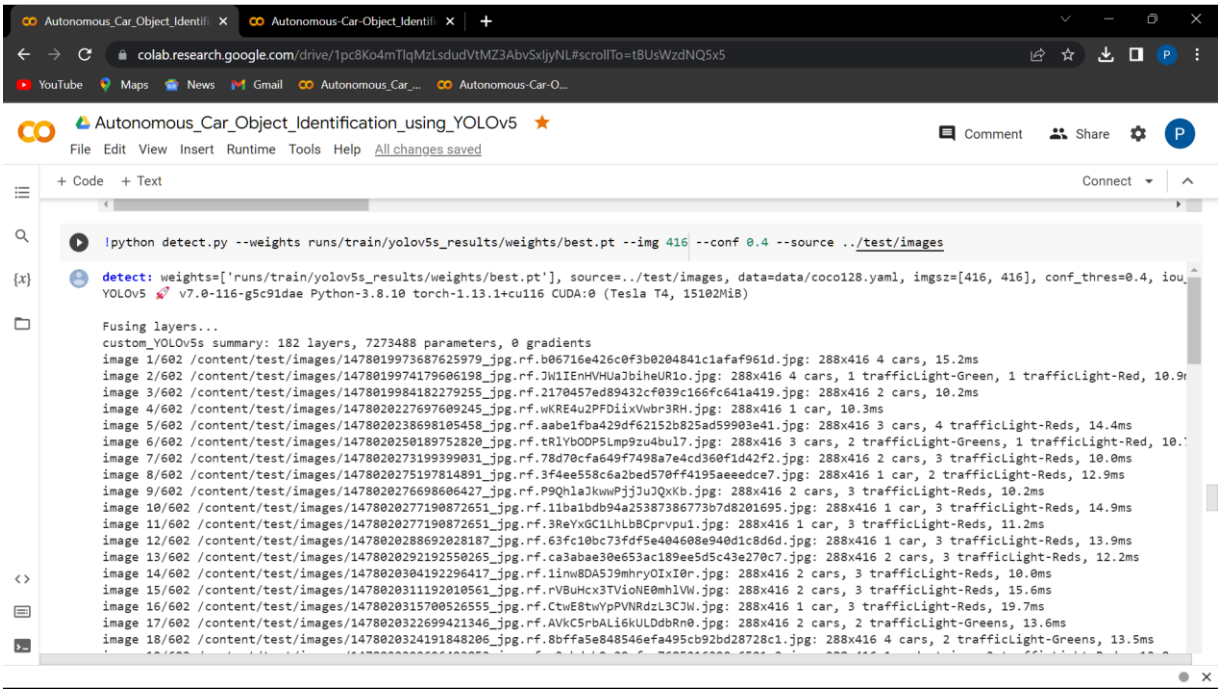

Autonomous_Car_Object_Identifi
Autonomous-Car-Object_Identifi
colab.research.google.com/drive/1pc8Ko4mTlqMzLsdudVtMZ3AbvSxljyNL#scrollTo=tBUsWzdNQ5x5
YouTube
Maps
News
Gmail
Autonomous_Car_...
Autonomous-Car-O...
Autonomous_Car_Object_Identification_using_YOLOv5
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
P
+ Code
+ Text
Connect
1919.0 479 600
1919 479 602
#extracting information from the roboflow file
%cat data.yaml
train: ../train/images
val: ../valid/images
nc: 11
names: ['biker', 'car', 'pedestrian', 'trafficLight', 'trafficLight-Green', 'trafficLight-GreenLeft', 'trafficLight-Red', 'trafficLight-RedLeft', 'trai
[ ] # define number of classes based on data.yaml
import yaml
with open("data.yaml", 'r') as stream:
num_classes = str(yaml.safe_load(stream)['nc'])
[ ] %cat /content/yolov5/models/yolov5s.yaml
YOLOv5 by Ultralytics, GPL-3.0 license
Parameters
nc: 80 # number of classes

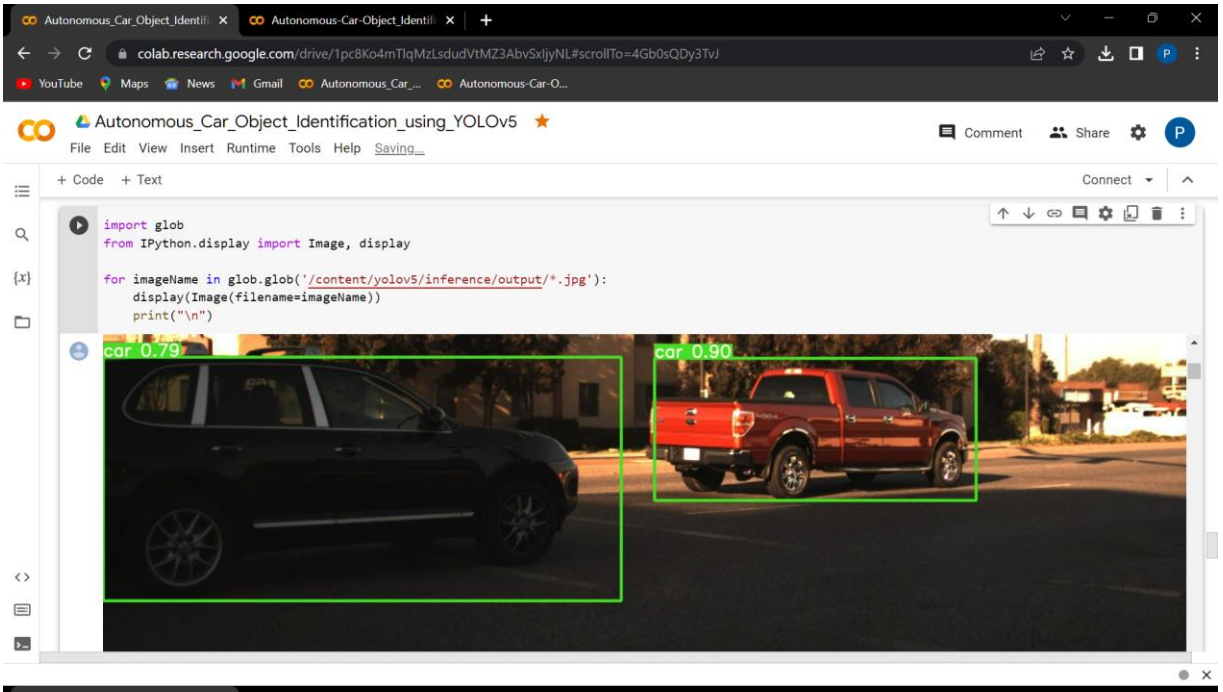
Autonomous_Car_Object_Identifi
Autonomous-Car-Object_Identifi
colab.research.google.com/drive/1pc8Ko4mTlqMzLsdudVtMZ3AbvSxljyNL#scrollTo=tBUsWzdNQ5x5
YouTube
Maps
News
Gmail
Autonomous_Car_...
Autonomous-Car-O...
Autonomous_Car_Object_Identification_using_YOLOv5
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
P
+ Code
+ Text
Connect
%cat /content/yolov5/models/yolov5s.yaml
YOLOv5 by Ultralytics, GPL-3.0 license
Parameters
nc: 80 # number of classes
depth_multiple: 0.33 # model depth multiple
width_multiple: 0.50 # layer channel multiple
anchors:
- [10,13, 16,30, 33,23] # P3/8
- [30,61, 62,45, 59,119] # P4/16
- [116,90, 156,198, 373,326] # P5/32
YOLOv5 v6.0 backbone
backbone:
[from, number, module, args]
[[-1, 1, Conv, [64, 6, 2, 2]], # 0-P1/2
[-1, 1, Conv, [128, 3, 2]], # 1-P2/4
[-1, 3, C3, [128]],
[-1, 1, Conv, [256, 3, 2]], # 3-P3/8
[-1, 6, C3, [256]],
[-1, 1, Conv, [512, 3, 2]], # 5-P4/16
[-1, 9, C3, [512]],
[-1, 1, Conv, [1024, 3, 2]], # 7-P5/32
[-1, 3, C3, [1024]],

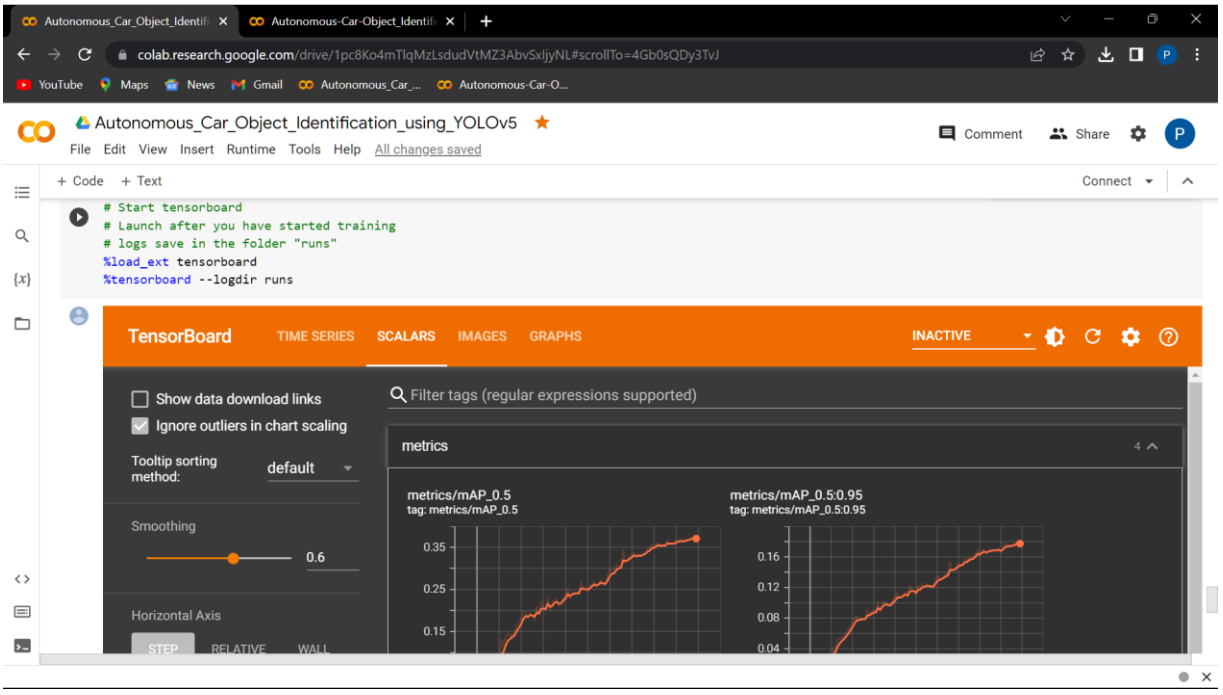

Autonomous_Car_Object_Identifi
Autonomous-Car-Object_Identifi
colab.research.google.com/drive/1pc8Ko4mTlqMzLsdudVtMZ3AbvSxIjyNL#scrollTo=tBUsWzdNQ5x5
YouTube
Maps
News
Gmail
Autonomous_Car_...
Autonomous-Car-O...
Autonomous_Car_Object_Identification_using_YOLOv5
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code
+ Text
Connect
[-1, 3, C3, [1024, False]], # 23 (P5/32-large)
[[17, 20, 23], 1, Detect, [nc, anchors]], # Detect(P3, P4, P5)
]
#customize iPython writefile so we can write variables
from IPython.core.magic import register_line_cell_magic
@register_line_cell_magic
def writetemplate(line,cell):
with open(line, 'w') as f:
f.write(cell.format(**globals()))
%%writetemplate /content/yolov5/models/custom_yolov5s.yaml
parameters
nc: {num_classes} # number of classes
depth_multiple: 0.33 # model depth multiple
width_multiple: 0.50 # layer channel multiple
anchors
anchors:
- [10,13, 16,30, 33,23] # P3/8
- [30,61, 62,45, 59,119] # P4/16

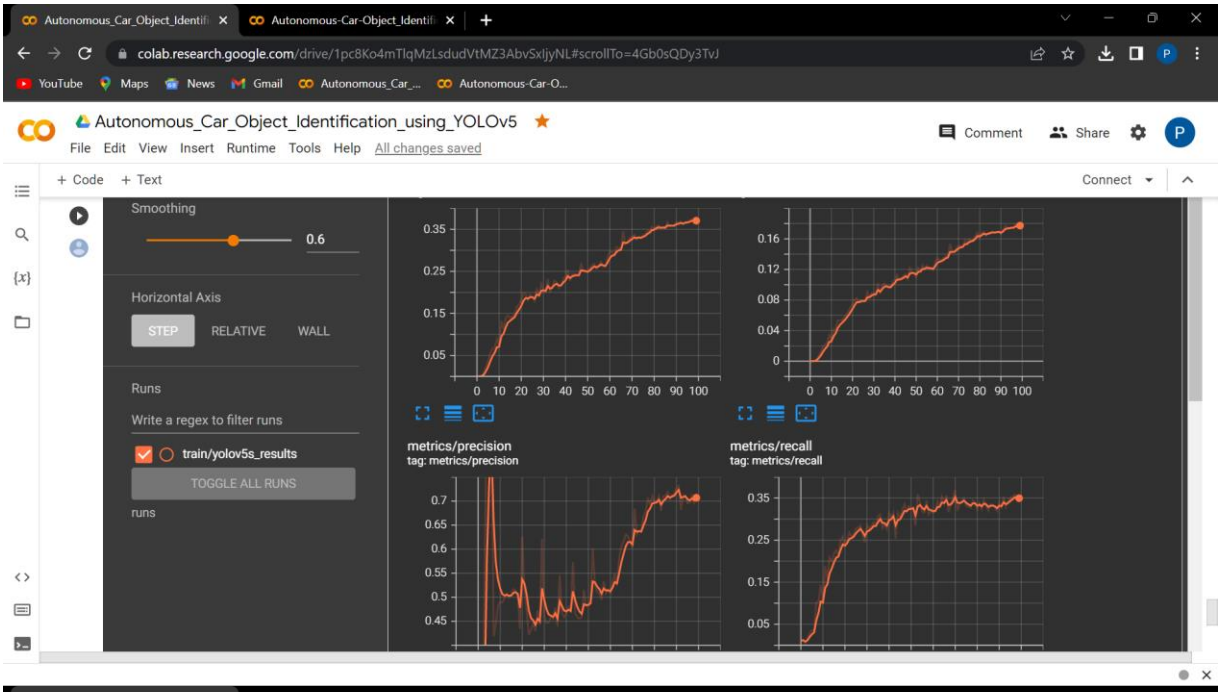

Autonomous_Car_Object_Identifi
Autonomous-Car-Object_Identifi
colab.research.google.com/drive/1pc8Ko4mTlqMzLsdudVtMZ3AbvSxIjyNL#scrollTo=tBUsWzdNQ5x5
YouTube
Maps
News
Gmail
Autonomous_Car_...
Autonomous-Car-O...
Autonomous_Car_Object_Identification_using_YOLOv5
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code
+ Text
Connect
%%writetemplate /content/yolov5/models/custom_yolov5s.yaml
parameters
nc: {num_classes} # number of classes
depth_multiple: 0.33 # model depth multiple
width_multiple: 0.50 # layer channel multiple
anchors
anchors:
- [10,13, 16,30, 33,23] # P3/8
- [30,61, 62,45, 59,119] # P4/16
- [116,90, 156,198, 373,326] # P5/32
YOLOv5 backbone
backbone:
[from, number, module, args]
[[-1, 1, Focus, [64, 3]], # 0-P1/2
[-1, 1, Conv, [128, 3, 2]], # 1-P2/4
[-1, 3, BottleneckCSP, [128]],
[-1, 1, Conv, [256, 3, 2]], # 3-P3/8
[-1, 9, BottleneckCSP, [256]],
[-1, 1, Conv, [512, 3, 2]], # 5-P4/16
[-1, 9, BottleneckCSP, [512]],
[-1, 1, Conv, [1024, 3, 2]], # 7-P5/32

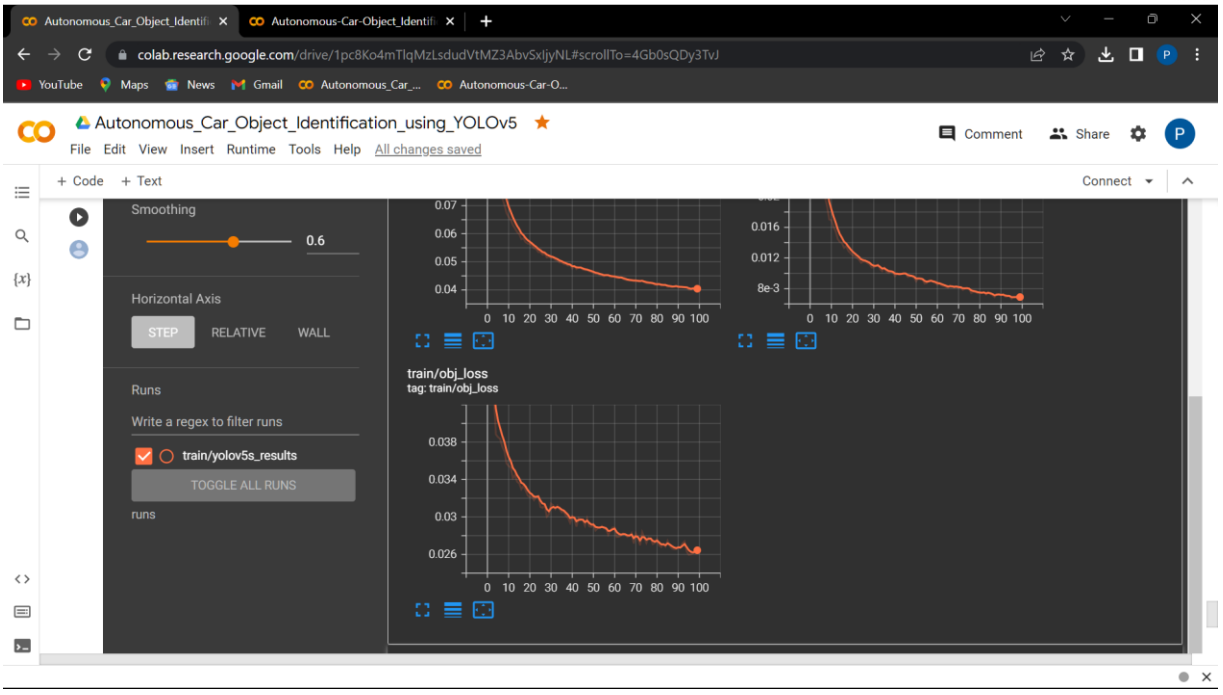

Autonomous_Car_Object_Identifi
Autonomous-Car-Object_Identifi
colab.research.google.com/drive/1pc8Ko4mTlqMzLsdudVtMZ3AbvSxljyNL#scrollTo=tBUsWzdNQ5x5
YouTube
Maps
News
Gmail
Autonomous_Car_...
Autonomous-Car-O...
Autonomous_Car_Object_Identification_using_YOLOv5
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code
+ Text
Connect
/content/yolov5
train: weights=, cfg=./models/custom_yolov5s.yaml, data=../data.yaml, hyp=data/hyps/hyp.scratch-low.yaml, epochs=100, batch_size=16, imgsz=416, rect
github: up to date with https://github.com/ultralytics/yolov5
YOLOv5 v7.0-116-g5c91dae Python-3.8.10 torch-1.13.1+cu116 CUDA:0 (Tesla T4, 15102MiB)
hyperparameters: lr0=0.01, lrf=0.01, momentum=0.937, weight_decay=0.0005, warmup_epochs=3.0, warmup_momentum=0.8, warmup_bias_lr=0.1, box=0.05, cls=
ClearML: run 'pip install clearml' to automatically track, visualize and remotely train YOLOv5 in ClearML
Comet: run 'pip install comet_ml' to automatically track and visualize YOLOv5 runs in Comet
TensorBoard: Start with 'tensorboard --logdir runs/train', view at http://localhost:6006/
2023-03-05 10:03:03.342878: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlow binary is optimized with oneAPI Deep Neural Network
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
2023-03-05 10:03:06.301833: W tensorflow/compiler/xla/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic library 'libnvinfer.
2023-03-05 10:03:06.302337: W tensorflow/compiler/xla/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic library 'libnvinfer_
2023-03-05 10:03:06.302370: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Cannot dlopen some TensorRT libraries. If you wo
Downloading https://ultralytics.com/assets/Arial.ttf to /root/.config/Ultralytics/Arial.ttf...
100% 755k/755k [00:00<00:00, 45.9MB/s]
from n params module arguments
0 -1 1 3520 models.common.Focus [3, 32, 3]
1 -1 1 18560 models.common.Conv [32, 64, 3, 2]
2 -1 1 19904 models.common.BottleneckCSP [64, 64, 1]
3 -1 1 73984 models.common.Conv [64, 128, 3, 2]
4 -1 3 161152 models.common.BottleneckCSP [128, 128, 3]
5 -1 1 295424 models.common.Conv [128, 256, 3, 2]
6 -1 3 641792 models.common.BottleneckCSP [256, 256, 3]
7 -1 1 1180672 models.common.Conv [256, 512, 3, 2]
8 -1 1 656896 models.common.SPP [512, 512, [5, 9, 13]]

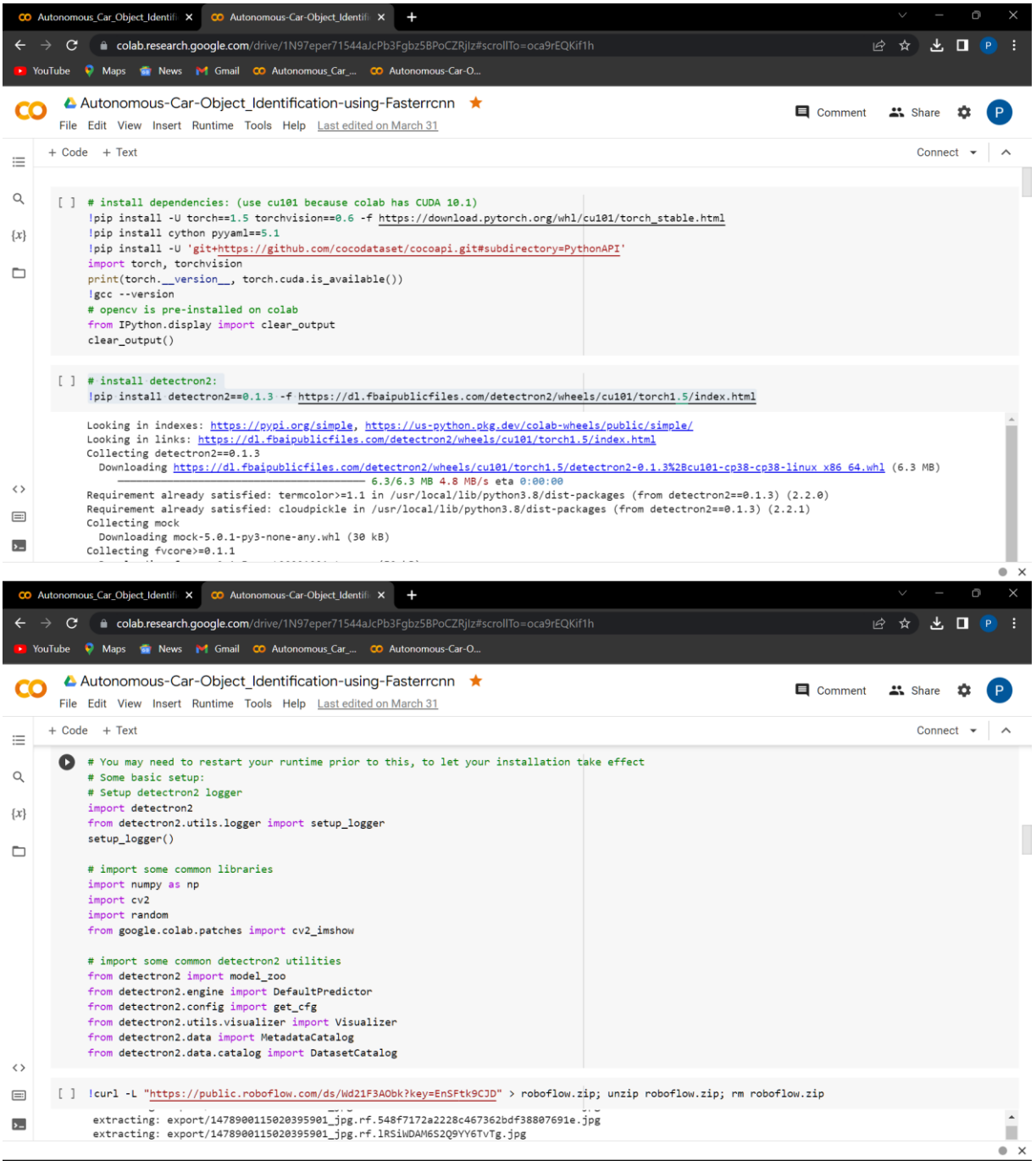

Autonomous_Car_Object_Identifi
Autonomous-Car-Object_Identifi
colab.research.google.com/drive/1pc8Ko4mTlqMzLsdudVtMZ3AbvSxljyNL#scrollTo=tBUsWzdNQ5x5
YouTube
Maps
News
Gmail
Autonomous_Car_...
Autonomous-Car-O...
Autonomous_Car_Object_Identification_using_YOLOv5
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code
+ Text
Connect
!python detect.py --weights runs/train/yolov5s_results/weights/best.pt --img 416 --conf 0.4 --source ../test/images
detect: weights=['runs/train/yolov5s_results/weights/best.pt'], source=../test/images, data=data/coco128.yaml, imgsz=[416, 416], conf_thres=0.4, iou
YOLOv5 v7.0-116-g5c91dae Python-3.8.10 torch-1.13.1+cu116 CUDA:0 (Tesla T4, 15102MiB)
Fusing layers...
custom_YOLOv5s summary: 182 layers, 7273488 parameters, 0 gradients
image 1/602 /content/test/images/1478019973687625979_jpg.rf.b06716e426c0f3b0204841c1afaf961d.jpg: 288x416 4 cars, 15.2ms
image 2/602 /content/test/images/1478019974179606198_jpg.rf.JW1IEnHVHUaJbiheUR1o.jpg: 288x416 4 cars, 1 trafficLight-Green, 1 trafficLight-Red, 10.9
image 3/602 /content/test/images/1478019984182279255_jpg.rf.2170457ed89432cf039c166fc641a419.jpg: 288x416 2 cars, 10.2ms
image 4/602 /content/test/images/1478020227697609245_jpg.rf.wKRE4u2PFDiixVwbr3RH.jpg: 288x416 1 car, 10.3ms
image 5/602 /content/test/images/1478020238698105458_jpg.rf.aabe1fba429df62152b825ad59903e41.jpg: 288x416 3 cars, 4 trafficLight-Reds, 14.4ms
image 6/602 /content/test/images/1478020250189752820_jpg.rf.tRlYbODP5Lmp9zu4bul7.jpg: 288x416 3 cars, 2 trafficLight-Greens, 1 trafficLight-Red, 10.
image 7/602 /content/test/images/1478020273199399031_jpg.rf.78d70cfa649f7498a7e4cd360f1d42f2.jpg: 288x416 2 cars, 3 trafficLight-Reds, 10.0ms
image 8/602 /content/test/images/1478020275197814891_jpg.rf.3f4ee558c6a2bed570ff4195aeeedce7.jpg: 288x416 1 car, 2 trafficLight-Reds, 12.9ms
image 9/602 /content/test/images/1478020276698606427_jpg.rf.P9QhlaJkwwPjjJuJQxKb.jpg: 288x416 2 cars, 3 trafficLight-Reds, 10.2ms
image 10/602 /content/test/images/1478020277190872651_jpg.rf.11ba1bdb94a25387386773b7d8201695.jpg: 288x416 1 car, 3 trafficLight-Reds, 14.9ms
image 11/602 /content/test/images/1478020277190872651_jpg.rf.3ReYxGC1LhLbBCprvpu1.jpg: 288x416 1 car, 3 trafficLight-Reds, 11.2ms
image 12/602 /content/test/images/1478020288692028187_jpg.rf.63fc10bc73fdf5e404608e940d1c8d6d.jpg: 288x416 1 car, 3 trafficLight-Reds, 13.9ms
image 13/602 /content/test/images/1478020292192550265_jpg.rf.ca3abae30e653ac189ee5d5c43e270c7.jpg: 288x416 2 cars, 3 trafficLight-Reds, 12.2ms
image 14/602 /content/test/images/1478020304192296417_jpg.rf.1inw8DA5J9mhryOIxI0r.jpg: 288x416 2 cars, 3 trafficLight-Reds, 10.0ms
image 15/602 /content/test/images/1478020311192010561_jpg.rf.rVBuHcx3TViоNE0mhlVW.jpg: 288x416 2 cars, 3 trafficLight-Reds, 15.6ms
image 16/602 /content/test/images/1478020315700526555_jpg.rf.CtwE8twYpPVNRdzL3CJW.jpg: 288x416 1 car, 3 trafficLight-Reds, 19.7ms
image 17/602 /content/test/images/1478020322699421346_jpg.rf.AVkC5rbALi6kULDdbRn0.jpg: 288x416 2 cars, 2 trafficLight-Greens, 13.6ms
image 18/602 /content/test/images/1478020324191848206_jpg.rf.8bffa5e848546efa495cb92bd28728c1.jpg: 288x416 4 cars, 2 trafficLight-Greens, 13.5ms


Autonomous_Car_Object_Identification_using_YOLOv5
File Edit View Insert Runtime Tools Help Saving...
Comment
Share
+ Code + Text
Connect
import glob
from IPython.display import Image, display
for imageName in glob.glob('/content/yolov5/inference/output/*.jpg'):
display(Image(filename=imageName))
print("\n")
car 0.79
car 0.90

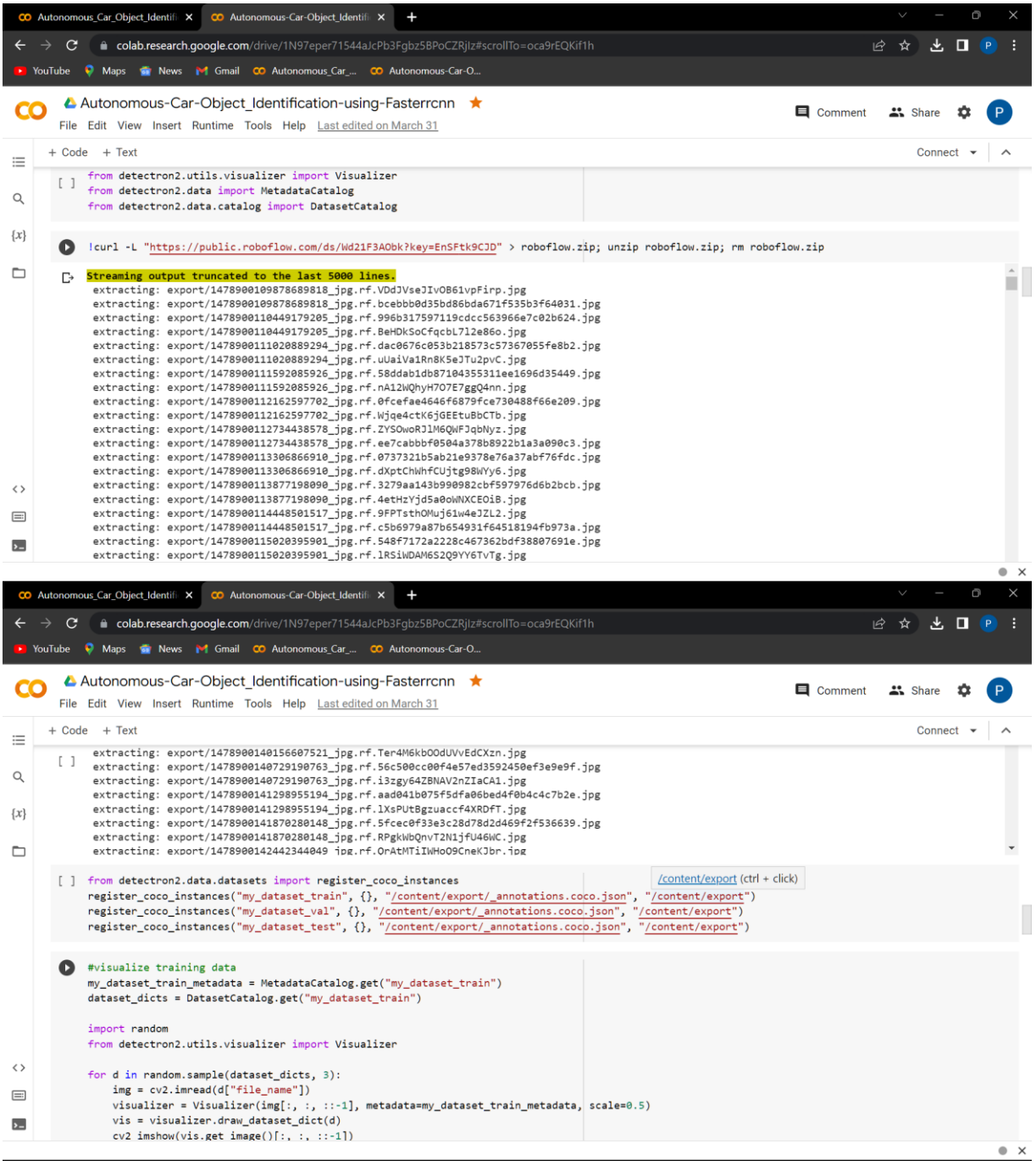

Autonomous_Car_Object_Identification_using_YOLOv5
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
Connect
Start tensorboard
Launch after you have started training
logs save in the folder "runs"
%load_ext tensorboard
%tensorboard --logdir runs
TensorBoard
TIME SERIES
SCALARS
IMAGES
GRAPHS
INACTIVE
Show data download links
Ignore outliers in chart scaling
Tooltip sorting method:
default
Smoothing
0.6
Horizontal Axis
STEP
RELATIVE
WALL
Filter tags (regular expressions supported)
metrics
metrics/mAP_0.5
tag: metrics/mAP_0.5
metrics/mAP_0.5:0.95
tag: metrics/mAP_0.5:0.95


Autonomous_Car_Object_Identification_using_YOLOv5
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code
+ Text
Connect
Smoothing
0.6
Horizontal Axis
STEP
RELATIVE
WALL
Runs
Write a regex to filter runs
train/yolov5s_results
TOGGLE ALL RUNS
runs
metrics/precision
tag: metrics/precision
metrics/recall
tag: metrics/recall


Autonomous_Car_Object_Identification_using_YOLOv5
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code
+ Text
Connect
Smoothing
0.6
Horizontal Axis
STEP
RELATIVE
WALL
Runs
Write a regex to filter runs
train/yolov5s_results
TOGGLE ALL RUNS
runs
train/obj_loss
tag: train/obj_loss


Autonomous_Car_Object_Identifi
Autonomous-Car-Object_Identifi
colab.research.google.com/drive/1N97eper71544aJcPb3Fgbz5BPoCZRjlz#scrollTo=oca9rEQKif1h
YouTube
Maps
News
Gmail
Autonomous_Car_...
Autonomous-Car-O...
Autonomous-Car-Object_Identification-using-Fasterrcnn
File Edit View Insert Runtime Tools Help Last edited on March 31
Comment
Share
+ Code + Text
Connect
[ ] # install dependencies: (use cu101 because colab has CUDA 10.1)
!pip install -U torch==1.5 torchvision==0.6 -f https://download.pytorch.org/whl/cu101/torch_stable.html
!pip install cython pyyaml==5.1
!pip install -U 'git+https://github.com/cocodataset/cocoapi.git#subdirectory=PythonAPI'
import torch, torchvision
print(torch.__version__, torch.cuda.is_available())
!gcc --version
# opencv is pre-installed on colab
from IPython.display import clear_output
clear_output()
[ ] # install detectron2:
!pip install detectron2==0.1.3 -f https://dl.fbaipublicfiles.com/detectron2/wheels/cu101/torch1.5/index.html
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Looking in links: https://dl.fbaipublicfiles.com/detectron2/wheels/cu101/torch1.5/index.html
Collecting detectron2==0.1.3
Downloading https://dl.fbaipublicfiles.com/detectron2/wheels/cu101/torch1.5/detectron2-0.1.3%2Bcu101-cp38-cp38-linux_x86_64.whl (6.3 MB)
6.3/6.3 MB 4.8 MB/s eta 0:00:00
Requirement already satisfied: termcolor>=1.1 in /usr/local/lib/python3.8/dist-packages (from detectron2==0.1.3) (2.2.0)
Requirement already satisfied: cloudpickle in /usr/local/lib/python3.8/dist-packages (from detectron2==0.1.3) (2.2.1)
Collecting mock
Downloading mock-5.0.1-py3-none-any.whl (30 kB)
Collecting fvcore>=0.1.1

Autonomous_Car_Object_Identifi
Autonomous-Car-Object_Identifi
colab.research.google.com/drive/1N97eper71544aJcPb3Fgbz5BPoCZRjlz#scrollTo=oca9rEQKif1h
YouTube
Maps
News
Gmail
Autonomous_Car_...
Autonomous-Car-O...
Autonomous-Car-Object_Identification-using-Fasterrcnn
File Edit View Insert Runtime Tools Help Last edited on March 31
Comment
Share
+ Code + Text
Connect
# You may need to restart your runtime prior to this, to let your installation take effect
# Some basic setup:
# Setup detectron2 logger
import detectron2
from detectron2.utils.logger import setup_logger
setup_logger()
# import some common libraries
import numpy as np
import cv2
import random
from google.colab.patches import cv2_imshow
# import some common detectron2 utilities
from detectron2 import model_zoo
from detectron2.engine import DefaultPredictor
from detectron2.config import get_cfg
from detectron2.utils.visualizer import Visualizer
from detectron2.data import MetadataCatalog
from detectron2.data.catalog import DatasetCatalog
[ ] !curl -L "https://public.roboflow.com/ds/Wd21F3AObk?key=EnSFtk9CJD" > roboflow.zip; unzip roboflow.zip; rm roboflow.zip
extracting: export/1478900115020395901_jpg.rf.548f7172a2228c467362bdf38807691e.jpg
extracting: export/1478900115020395901_jpg.rf.lRSiWDAM6S2Q9YY6TvTg.jpg


Autonomous_Car_Object_Identifi
Autonomous-Car-Object_Identifi
colab.research.google.com/drive/1N97eper71544aJcPb3Fgbz5BPoCZRjlz#scrollTo=oca9rEQKif1h
YouTube
Maps
News
Gmail
Autonomous_Car_...
Autonomous-Car-O...
Autonomous-Car-Object_Identification-using-Fasterrcnn
File Edit View Insert Runtime Tools Help Last edited on March 31
Comment
Share
+ Code + Text
Connect
from detectron2.utils.visualizer import Visualizer
from detectron2.data import MetadataCatalog
from detectron2.data.catalog import DatasetCatalog
!curl -L "https://public.roboflow.com/ds/Wd21F3AObk?key=EnSFtk9CJD" > roboflow.zip; unzip roboflow.zip; rm roboflow.zip
Streaming output truncated to the last 5000 lines.
extracting: export/1478900109878689818_jpg.rf.VDdJVseJIvOB61vpFirp.jpg
extracting: export/1478900109878689818_jpg.rf.bcebbb0d35bd86bda671f535b3f64031.jpg
extracting: export/1478900110449179205_jpg.rf.996b317597119cdcc563966e7c02b624.jpg
extracting: export/1478900110449179205_jpg.rf.BeHDkSoCfqcbL7l2e86o.jpg
extracting: export/1478900111020889294_jpg.rf.dac0676c053b218573c57367055fe8b2.jpg
extracting: export/1478900111020889294_jpg.rf.uUaiVa1Rn8K5eJTu2pvC.jpg
extracting: export/1478900111592085926_jpg.rf.58ddab1db87104355311ee1696d35449.jpg
extracting: export/1478900111592085926_jpg.rf.nA12WQhyH7O7E7ggQ4nn.jpg
extracting: export/1478900112162597702_jpg.rf.0fcefae4646f6879fce730488f66e209.jpg
extracting: export/1478900112162597702_jpg.rf.Wjqe4ctK6jGEEtuBbCTb.jpg
extracting: export/1478900112734438578_jpg.rf.ZYSOwoRJlM6QWFJqbNyz.jpg
extracting: export/1478900112734438578_jpg.rf.ee7cabbbf0504a378b8922b1a3a090c3.jpg
extracting: export/1478900113306866910_jpg.rf.0737321b5ab21e9378e76a37abf76fdc.jpg
extracting: export/1478900113306866910_jpg.rf.dXptChWhfCUjtg98WYy6.jpg
extracting: export/1478900113877198090_jpg.rf.3279aa143b990982cbf597976d6b2bcb.jpg
extracting: export/1478900113877198090_jpg.rf.4etHzYjd5a0oWNXCEOiB.jpg
extracting: export/1478900114448501517_jpg.rf.9FPTsthOMuj61w4eJZL2.jpg
extracting: export/1478900114448501517_jpg.rf.c5b6979a87b654931f64518194fb973a.jpg
extracting: export/1478900115020395901_jpg.rf.548f7172a2228c467362bdf38807691e.jpg
extracting: export/1478900115020395901_jpg.rf.lRSiWDAM6S2Q9YY6TvTg.jpg

Autonomous_Car_Object_Identifi
Autonomous-Car-Object_Identifi
colab.research.google.com/drive/1N97eper71544aJcPb3Fgbz5BPoCZRjlz#scrollTo=oca9rEQKif1h
YouTube
Maps
News
Gmail
Autonomous_Car_...
Autonomous-Car-O...
Autonomous-Car-Object_Identification-using-Fasterrcnn
File Edit View Insert Runtime Tools Help Last edited on March 31
Comment
Share
+ Code + Text
Connect
extracting: export/1478900140156607521_jpg.rf.Ter4M6kbOOdUVvEdCXzn.jpg
extracting: export/1478900140729190763_jpg.rf.56c500cc00f4e57ed3592450ef3e9e9f.jpg
extracting: export/1478900140729190763_jpg.rf.i3zgy64ZBNAV2nZIaCA1.jpg
extracting: export/1478900141298955194_jpg.rf.aad041b075f5dfa06bed4f0b4c4c7b2e.jpg
extracting: export/1478900141298955194_jpg.rf.lXsPUtBgzuaccf4XRDfT.jpg
extracting: export/1478900141870280148_jpg.rf.5fcec0f33e3c28d78d2d469f2f536639.jpg
extracting: export/1478900141870280148_jpg.rf.RPgkWbQnvT2N1jfU46WC.jpg
extracting: export/1478900142442344049_jpg.rf.OrAtMTiIWHoO9CneKJbr.jpg
from detectron2.data.datasets import register_coco_instances
register_coco_instances("my_dataset_train", {}, "/content/export/_annotations.coco.json", "/content/export")
register_coco_instances("my_dataset_val", {}, "/content/export/_annotations.coco.json", "/content/export")
register_coco_instances("my_dataset_test", {}, "/content/export/_annotations.coco.json", "/content/export")
/content/export (ctrl + click)
#visualize training data
my_dataset_train_metadata = MetadataCatalog.get("my_dataset_train")
dataset_dicts = DatasetCatalog.get("my_dataset_train")
import random
from detectron2.utils.visualizer import Visualizer
for d in random.sample(dataset_dicts, 3):
img = cv2.imread(d["file_name"])
visualizer = Visualizer(img[:, :, ::-1], metadata=my_dataset_train_metadata, scale=0.5)
vis = visualizer.draw_dataset_dict(d)
cv2_imshow(vis.get_image()[:, :, ::-1])

Autonomous_Car_Object_Identifi | Autonomous-Car-Object_Identifi

colab.research.google.com/drive/1N97eper71544aJcPb3Fgbz5BPoCZRjlz#scrollTo=oca9rEQKif1h

YouTube Maps News Gmail Autonomous_Car_... Autonomous-Car-O...

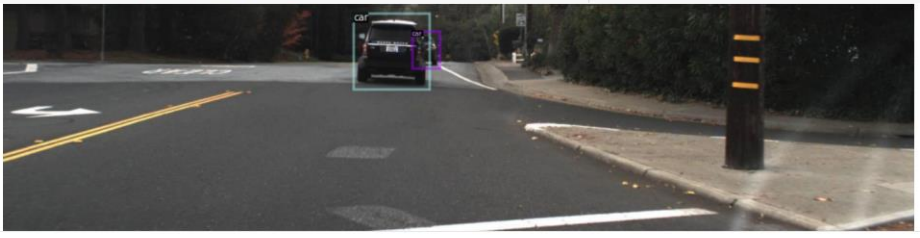
Autonomous-Car-Object_Identification-using-Fasterrcnn

File Edit View Insert Runtime Tools Help Last edited on March 31

Comment Share

+ Code + Text Connect

```
#visualize training data
my_dataset_train_metadata = MetadataCatalog.get("my_dataset_train")
dataset_dicts = DatasetCatalog.get("my_dataset_train")

import random
from detectron2.utils.visualizer import Visualizer

for d in random.sample(dataset_dicts, 3):
    img = cv2.imread(d["file_name"])
    visualizer = Visualizer(img[:, :, ::-1], metadata=my_dataset_train_metadata, scale=0.5)
    vis = visualizer.draw_dataset_dict(d)
    cv2_imshow(vis.get_image()[:, :, ::-1])
```



Autonomous_Car_Object_Identifi | Autonomous-Car-Object_Identifi

colab.research.google.com/drive/1N97eper71544aJcPb3Fgbz5BPoCZRjlz#scrollTo=oca9rEQKif1h

YouTube Maps News Gmail Autonomous_Car_... Autonomous-Car-O...

Autonomous-Car-Object_Identification-using-Fasterrcnn

File Edit View Insert Runtime Tools Help Last edited on March 31

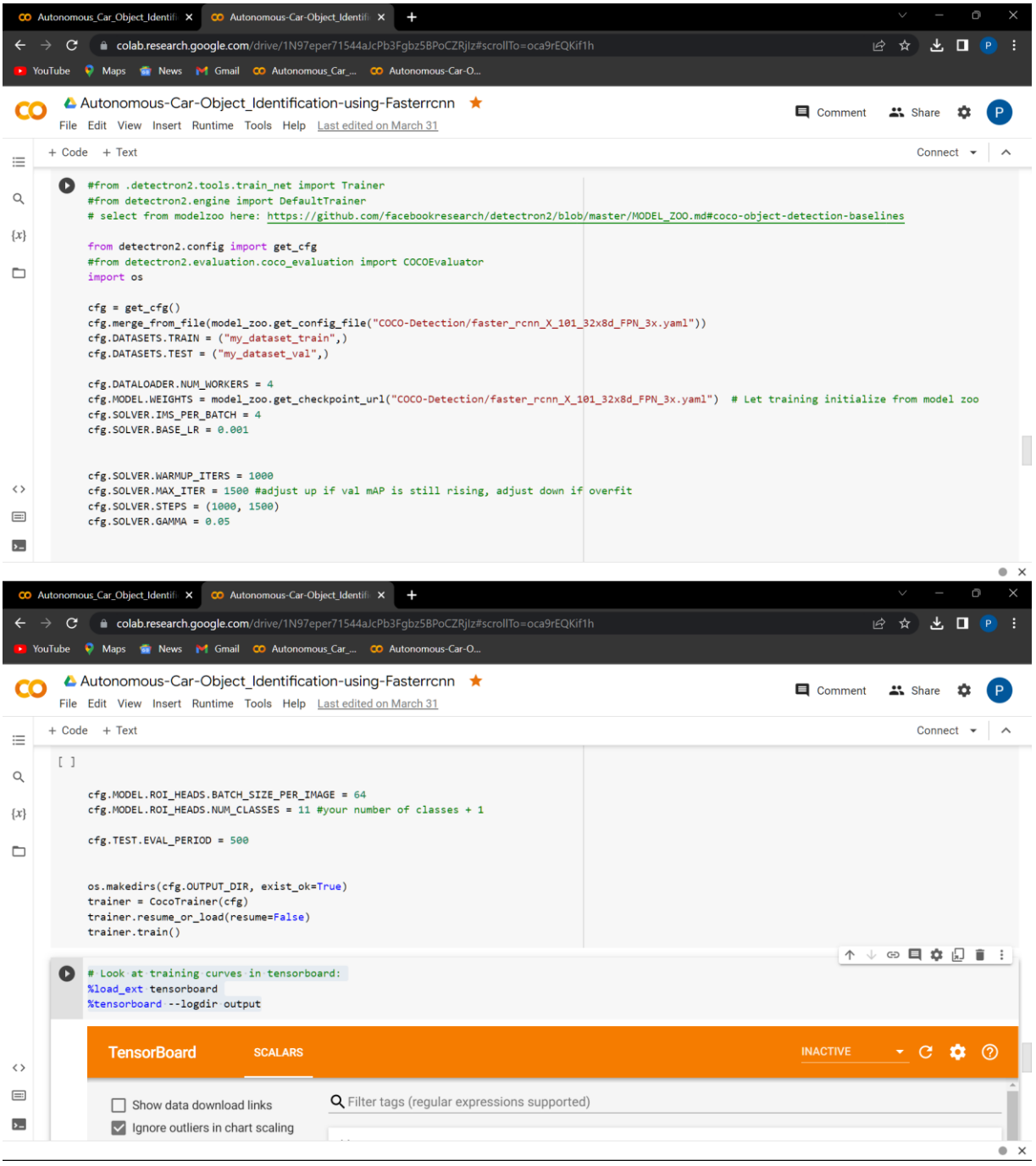
Comment Share

+ Code + Text Connect

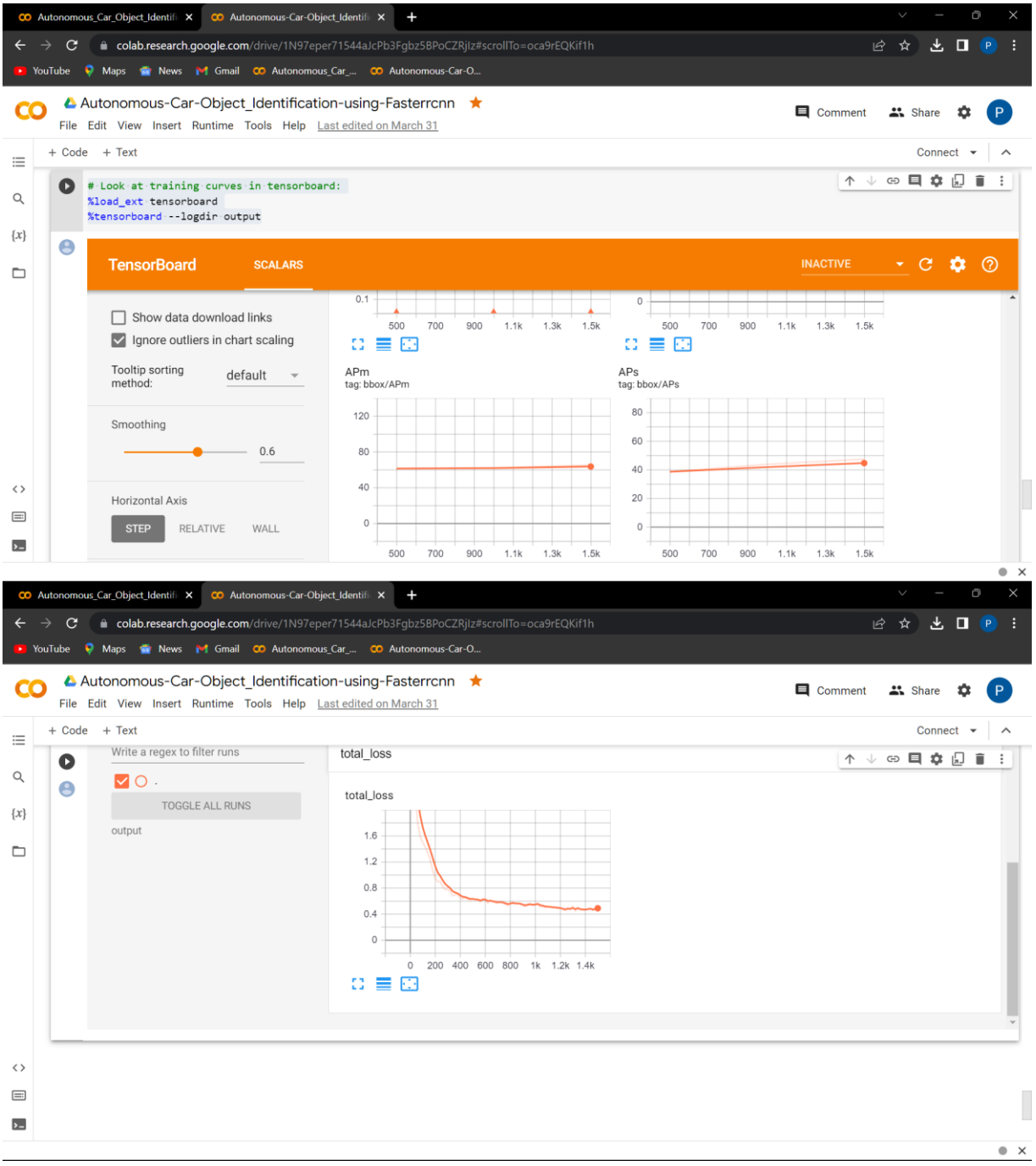
[ ]

```
[ ] #We are importing our own Trainer Module here to use the COCO validation evaluation during training. Otherwise no validation eval occurs.

    from detectron2.engine import DefaultTrainer
    from detectron2.evaluation import COCOEvaluator

    class CocoTrainer(DefaultTrainer):

      @classmethod
      def build_evaluator(cls, cfg, dataset_name, output_folder=None):

        if output_folder is None:
            os.makedirs("coco_eval", exist_ok=True)
            output_folder = "coco_eval"

        return COCOEvaluator(dataset_name, cfg, False, output_folder)
```

```
[ ] #from .detectron2.tools.train_net import Trainer
    #from detectron2.engine import DefaultTrainer
    # select from modelzoo here: https://github.com/facebookresearch/detectron2/blob/master/MODEL_ZOO.md#coco-object-detection-baselines
```


Autonomous_Car_Object_Identifi
Autonomous-Car-Object_Identifi
colab.research.google.com/drive/1N97eper71544aJcPb3Fgbz5BPoCZRjlz#scrollTo=oca9rEQKif1h
YouTube
Maps
News
Gmail
Autonomous_Car_...
Autonomous-Car-O...
Autonomous-Car-Object_Identification-using-Fasterrcnn
File Edit View Insert Runtime Tools Help Last edited on March 31
Comment
Share
+ Code
+ Text
Connect
#from .detectron2.tools.train_net import Trainer
#from detectron2.engine import DefaultTrainer
# select from modelzoo here: https://github.com/facebookresearch/detectron2/blob/master/MODEL_ZOO.md#coco-object-detection-baselines
from detectron2.config import get_cfg
#from detectron2.evaluation.coco_evaluation import COCOEvaluator
import os
cfg = get_cfg()
cfg.merge_from_file(model_zoo.get_config_file("COCO-Detection/faster_rcnn_X_101_32x8d_FPN_3x.yaml"))
cfg.DATASETS.TRAIN = ("my_dataset_train",)
cfg.DATASETS.TEST = ("my_dataset_val",)
cfg.DATALOADER.NUM_WORKERS = 4
cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url("COCO-Detection/faster_rcnn_X_101_32x8d_FPN_3x.yaml") # Let training initialize from model zoo
cfg.SOLVER.IMS_PER_BATCH = 4
cfg.SOLVER.BASE_LR = 0.001
cfg.SOLVER.WARMUP_ITERS = 1000
cfg.SOLVER.MAX_ITER = 1500 #adjust up if val mAP is still rising, adjust down if overfit
cfg.SOLVER.STEPS = (1000, 1500)
cfg.SOLVER.GAMMA = 0.05

Autonomous_Car_Object_Identifi
Autonomous-Car-Object_Identifi
colab.research.google.com/drive/1N97eper71544aJcPb3Fgbz5BPoCZRjlz#scrollTo=oca9rEQKif1h
YouTube
Maps
News
Gmail
Autonomous_Car_...
Autonomous-Car-O...
Autonomous-Car-Object_Identification-using-Fasterrcnn
File Edit View Insert Runtime Tools Help Last edited on March 31
Comment
Share
+ Code
+ Text
Connect
[ ]
cfg.MODEL.ROI_HEADS.BATCH_SIZE_PER_IMAGE = 64
cfg.MODEL.ROI_HEADS.NUM_CLASSES = 11 #your number of classes + 1
cfg.TEST.EVAL_PERIOD = 500
os.makedirs(cfg.OUTPUT_DIR, exist_ok=True)
trainer = CocoTrainer(cfg)
trainer.resume_or_load(resume=False)
trainer.train()
# Look at training curves in tensorboard:
%load_ext tensorboard
%tensorboard --logdir output
TensorBoard
SCALARS
INACTIVE
Show data download links
Ignore outliers in chart scaling
Filter tags (regular expressions supported)



RESEARCH PAPER: