

Semantic Segmentation Based Object Detection for Autonomous Driving

by

HARSH PRAJAPATI
202111074

A Thesis Submitted in Partial Fulfilment of the Requirements for the Degree of

MASTER OF TECHNOLOGY
in
INFORMATION AND COMMUNICATION TECHNOLOGY
to

DHIRUBHAI AMBANI INSTITUTE OF INFORMATION AND COMMUNICATION TECHNOLOGY



July, 2023

Declaration

I hereby declare that

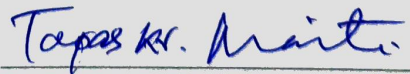
- i) the thesis comprises of my original work towards the degree of Master of Technology in Information and Communication Technology at Dhirubhai Ambani Institute of Information and Communication Technology and has not been submitted elsewhere for a degree,
- ii) due acknowledgment has been made in the text to all the reference material used.



Harsh Prajapati

Certificate

This is to certify that the thesis work entitled "Semantic Segmentation Based Object Detection for Autonomous Driving" has been carried out by Harsh Prajapati for the degree of Master of Technology in Information and Communication Technology at *Dhirubhai Ambani Institute of Information and Communication Technology* under my supervision.



Prof. Tapas Kumar Maiti
Thesis Supervisor

Acknowledgments

I would like to express my heartfelt gratitude to Prof. Tapas Kumar Maiti, my esteemed supervisor, whose profound expertise and unwavering commitment to scholarly excellence have served as an invaluable compass throughout my thesis. His profound insights, constructive critique, and steadfast encouragement have significantly influenced the trajectory and caliber of my research. I am genuinely appreciative of his mentorship and the opportunities he provided me to evolve as a scholar.

I would also like to extend my sincere appreciation to Mr. Aditya Bhope, a fellow scholar, for his support. His readiness to extend a helping hand and his substantial contributions have enriched my understanding and significantly elevated the outcomes of this research. I am grateful for the intellectual synergy, I share and the bond of camaraderie, I have cultivated throughout this transformative journey.

Furthermore, I would like to convey my deep appreciation to my dear friend, Vraj. His unwavering belief in my abilities, constant encouragement during challenging times, and unique capacity to offer fresh insights have been a continuous source of inspiration. His support and friendship have played a pivotal role in keeping me motivated and reinforcing the importance of balance and well-being during this demanding process. I am genuinely thankful for his presence in my life and the profound impact he had on my personal and academic growth.

Contents

Abstract	vi
List of Principal Symbols and Acronyms	vi
List of Tables	vii
List of Figures	viii
1 Introduction	1
1.1 About an Artificial Intelligent System	1
1.2 Semantic Segmentation and Challenges	3
1.3 Application of Semantic Segmentation	4
1.4 Semantic Segmentation for Autonomous Driving	4
2 Literature Survey	6
2.1 Robot Operating System (ROS)	6
2.1.1 Use of ROS	7
2.1.2 ROS Computation Graph Model	7
2.1.3 Nodes	7
2.1.4 Master	7
2.1.5 Topics	8
2.1.6 Services	8
2.1.7 Parameter Server	9
2.2 Libraries and Tools	9
2.2.1 Gazebo Simulator	9
2.2.2 Point Cloud Library	9
2.2.3 OpenCV	10
2.2.4 ROS Bag	10
2.2.5 Rviz	11
2.2.6 Catkin	11
2.2.7 ROS bash	11

2.2.8	ROS launch	12
2.2.9	ROS Versions	12
2.3	Object Detection Model	13
3	YOLO Architecture	16
3.1	Introduction	16
3.2	Versions	17
3.2.1	YOLOv1	17
3.2.2	YOLOv2	18
3.2.3	YOLOv3	19
3.2.4	YOLOv4	20
3.3	Architecture	21
3.4	Dataset	24
3.4.1	Berkeley Dataset	24
3.4.2	Labelling Tool	25
3.4.3	Cityscapes Dataset	26
3.4.4	Our Dataset	28
3.5	Verification of YOLO Model	29
3.5.1	Mode Variants	29
3.5.2	Training	30
3.5.3	Results	32
3.6	Limitation	39
4	Semantic Segmentation	40
4.1	Neural Architecture Search	41
4.2	Teacher/Student Co-searching For Knowledge Distillation	42
4.3	FasterSeg	43
4.4	Optimizing Search Space for Efficient Multi-Resolution Branching	43
4.4.1	Searchable Multi-Resolution Branches	44
4.4.2	Selecting Optimal Operators for Enhanced Receptive Field Coverage	44
4.4.3	Searchable Super kernel For Expansion Ratios	45
4.5	Regularized Latency Optimization with Finer Granularity	45
4.6	Verifications	46
4.6.1	Architecture Search	46
4.6.2	Exploring the Effectiveness of Multi-Resolution Search Space and Collaborative Search	47
4.6.3	Results	49

5	Real-Time Experiments	53
5.1	Edge Device	53
5.2	Use Cases	54
5.3	Demonstration	56
6	Conclusions	58
	References	59

Abstract

This research focuses on solving the autonomous driving problem which is necessary to fulfill the increasing demand of autonomous systems in today's world. The key aspect in addressing this challenge is the real-time identification and recognition of objects within the driving environment. To accomplish this, we employ the semantic segmentation technique, integrating computer vision, machine learning, deep learning, the PyTorch framework, image processing, and the robot operating system (ROS). Our approach involves creating an experimental setup using an edge device, specifically a Raspberry Pi, in conjunction with the ROS framework. By deploying a deep learning model on the edge device, we aim to build a robust and efficient autonomous system that can accurately identify and recognize objects in real time.

List of Tables

2.1	Depicted ROS versions published in numerous years.	12
3.1	Overview of YOLOv5 Different Model	30
3.2	Information of Hyper-parameters	31
3.3	Model Performance Summary on Validation Dataset	31
4.1	Studies of Numerous Search and Training Strategies	47
4.2	Comparison of the models	48

List of Figures

1.1	Overview of the Thesis	2
2.1	DA Logo	8
3.1	Illustrates an overview of YOLO architectures.	23
3.2	Berkeley Deep Drive Dataset(1)	25
3.3	Berkeley Deep Drive Dataset(2)	25
3.4	Labellmg GUI tool.	26
3.5	Image from img8bit Cityscapes	27
3.6	Image from gtFine Cityscapes	28
3.7	Own Dataset(1)	29
3.8	Own Dataset(2)	29
3.9	Confusion Matrix	33
3.10	Precision-Recall Curve	34
3.11	F1-score Curve	35
3.12	Original Cityscape Dataset Image	35
3.13	Predicted Image of Cityscape Dataset	36
3.14	Original Image of Berkeley Dataset	37
3.15	Predicted Image of Berkeley Dataset	37
3.16	Original Image of Own Dataset	38
3.17	Predicted Image of Own Dataset	38
4.1	An overview of Neural Architecture Search	42
4.2	An overview of Multibranch Searching	43
4.3	Network Discovered by NAS Network	46
4.4	System Overview	48
4.5	Orginal Cityscape Image	49
4.6	Result Of Cityscape Image	50
4.7	Orginal Berkeley Image	50
4.8	Result Of Berkeley Image	51
4.9	Orginal Own Dataset Image	51

4.10	Result Of Own Dataset Image	52
5.1	Raspberry Pie Component	54
5.2	ROS Process for Experiment	57
5.3	Real time Experimental Setup	57

CHAPTER 1

Introduction

1.1 About an Artificial Intelligent System

An artificial intelligent (AI) system refers to a system that leverages artificial intelligence (AI) techniques to emulate or replicate human intelligence, showcasing intelligent behavior. It incorporates various AI technologies, including computer vision, natural language processing(NLP), machine learning(ML), and knowledge representation, to perform tasks that typically require human intelligence. By utilizing these AI capabilities, intelligent artificial systems can accomplish complex activities with efficiency and accuracy.

In this work, we focused on the autonomous driving bot problem. The advent of autonomous driving has transformed the automotive industry, accelerating the development of intelligent systems capable of navigating roads without human involvement. However, numerous critical obstacles must be addressed to achieve safe and dependable autonomous driving. Among these difficulties, object detection and recognition are critical in assuring the vehicle's ability to observe and interact with its surroundings. This thesis focuses on applying AI technology to tackle the segmentation of different objects in autonomous driving.

Autonomous driving systems rely mainly on their capacity to detect and recognize many items in their surroundings, such as pedestrians, vehicles, traffic signs, and traffic lights. Precise and dependable object detection is crucial for autonomous vehicles to make informed decisions, avoid obstacles, and protect passengers and other road users. We aim to improve object detection and recognition skills of a autonomous driving bot by using AI algorithms and approaches.

Our primary goal is to develop an algorithm that achieves high accuracy in real-time object detection while ensuring seamless deployment on edge device, mentioned detailed in chapters 4 and 5. We observed that many existing algorithms struggle to accurately identify objects under various conditions. This has motivated us to find the most suitable algorithm to address these challenges ef-

fectively. In our final stage, we integrated the camera, model, and edge device using the robot operating system (ROS) to create an autonomous driving bot which will explain in chapter 5. This comprehensive approach allows us to combine advanced object detection capabilities with the power of ROS, enabling the bot to navigate autonomously.

To address the object detection problem, we thoroughly explored numerous algorithms. After careful consideration, we concluded that computer vision algorithms offer the most promising solution, specifically semantic segmentation. The semantic segmentation technique excels in providing precise object boundaries, facilitating accurate identification. However, it is important to note that this approach demands higher computational resources due to its inherent complexity and detailed analysis. Despite the increased computational requirements, we firmly believe that the advantages of precise boundary detection outweigh the associated resource demands. An overview of the thesis work is presented in Fig. 1.1.

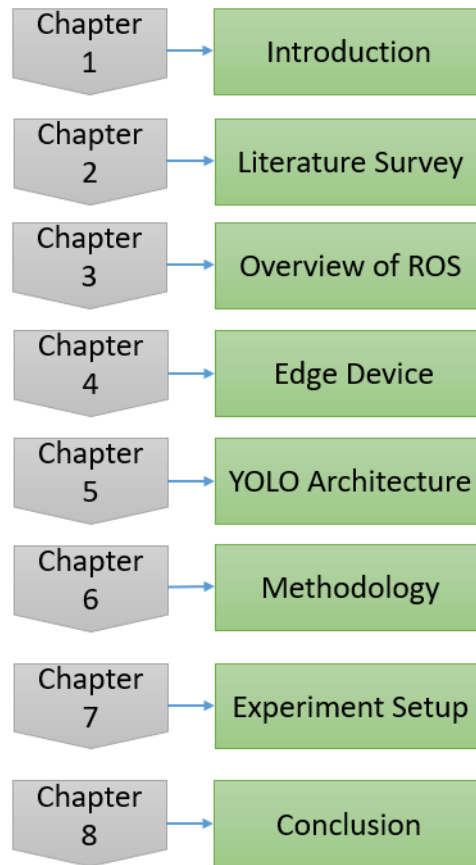


Figure 1.1: Chapter wise overview of the thesis work.

1.2 Semantic Segmentation and Challenges

Semantic segmentation is a task in computer vision that poses a significant challenge which involves partitioning an image into distinct regions or segments and assigning individual class labels to each pixel within the image [1]. Unlike conventional image segmentation methods that primarily focus on separating objects based on their boundaries, semantic segmentation aims to assign semantically meaningful labels to each pixel, enabling a comprehensive understanding of the image's content.

It enables computers to perceive and comprehend visual scenes at the pixel level, similar to how humans analyze and segment things within a picture. It is critical in various applications, including autonomous driving, medical imaging, robotics, augmented reality, and scene interpretation. Despite being a powerful computer vision approach, semantic segmentation has various problems. Some of the major issues in semantic segmentation are as follows:

Pixel-Level Accuracy: Achieving high pixel-level accuracy is a basic challenge in semantic segmentation. It necessitates reliably assigning the correct class label to each pixel in a picture, even when complicated backgrounds, occlusions, and confusing boundaries are present. Working with fine-grained features and little objects can be very difficult.

Class Imbalance: There is frequently a considerable class imbalance in many semantic segmentation datasets, meaning some classes may have significantly fewer pixels than others. This disparity can lead to biased learning and low performance in minority classes. The class imbalance must be addressed to achieve balanced and accurate semantic segmentation.

Boundary Ambiguity: When objects are adjacent or overlapping, the boundaries between various object instances or classes might be confusing. Separating neighboring items with similar visual properties can be difficult, resulting in misclassifications or fragmented segmentation findings.

Limited Training Data: Semantic segmentation models necessitate a significant quantity of labeled training data to acquire the complex relationship between input images and pixel-level annotations. However, acquiring pixel-level annotations for a large dataset can be expensive and time-consuming. Inadequate training data can result in overfitting or a lack of generalization in the model.

Computational Complexity: Semantic segmentation frequently includes deep neural networks analyzing high-resolution images, which can be computationally demanding. Real-time performance can be challenging for resource-constrained

devices or settings with severe latency constraints, necessitating optimization approaches, and efficient model structures.

Generalization to New Environments: Semantic segmentation methods trained on specific datasets may struggle to generalize to new and unknown environments. Models trained on a single dataset may perform poorly when applied to different domains or when lighting, weather, or camera views vary. It is a continuous issue to ensure the resilience and generalization of semantic segmentation models over a wide range of contexts.

1.3 Application of Semantic Segmentation

Semantic segmentation has numerous applications in a variety of disciplines. Here are some notable examples:

Autonomous Driving: Semantic segmentation is essential for autonomous cars to observe and interpret their surroundings. It aids in object identification, path planning, and obstacle avoidance by accurately segmenting the scene into multiple classes (e.g., roads, vehicles, pedestrians, traffic signs), boosting the safety and efficiency of self-driving automobiles.

Augmented Reality and Scene Understanding: Semantic segmentation helps with scene understanding by providing a detailed grasp of the objects and their spatial layout in an image or video. It utilizes in augmented reality apps to precisely overlay virtual objects on top of the actual world, ensuring perfect occlusion and interaction with the surroundings.

Medical Imaging: Semantic segmentation in medical imaging aids in correctly delineating and identifying various structures and organs in medical scans such as MRI, CT, or ultrasound. It is used to aid in diagnosis, treatment planning, and medical research by performing tasks such as tumor identification, organ segmentation, and tissue classification.

Urban Planning and Environmental Analysis: By recognizing and analyzing distinct features in aerial or satellite data, semantic segmentation can aid urban planning and environmental analysis. It helps with land-use classification, infrastructure mapping, vegetation analysis, and environmental change monitoring.

1.4 Semantic Segmentation for Autonomous Driving

Semantic segmentation plays a crucial role in object identification and recognition within the field of autonomous driving. It enables the identification and classifi-

cation of diverse road objects by assigning individual class labels to each pixel in an image. Within the context of autonomous driving, the following components are essential for semantic segmentation in object detection and recognition:

Object Localization: Semantic segmentation allows precise pixel-level localization of items in the scene. It enables autonomous vehicles to correctly determine the borders of various things, such as cars, pedestrians, bicycles, and barriers, by segmenting each object instance separately.

Fine-Grained Object Understanding: Semantic segmentation enables fine-grained object understanding by providing specific class labels to each pixel. This level of granularity enables autonomous vehicles to distinguish between different object categories, such as car types, traffic sign changes, or pedestrian attributes.

Handling Occlusions: Semantic segmentation aids in handling occlusions, which occur when other objects partially or entirely obscure objects. Autonomous cars can properly assess their positions and sizes by segmenting the viewable areas of occluded objects, assisting in collision avoidance and maneuvering decisions.

Real-Time Object Detection: Semantic segmentation techniques are specifically designed to operate in real time, providing efficient and continuous detection and recognition of objects. This capability is crucial in self-driving systems as it ensures timely perception and response, which are essential for safe and efficient operation.

CHAPTER 2

Literature Survey

2.1 Robot Operating System (ROS)

"ROS: An Open-Source Robot Operating System" by [2] is a seminal study in this field. This key work introduces ROS as an open-source software framework for robotics research and development. It describes ROS's architecture, communication processes, and essential features, emphasizing its benefits in fostering collaboration and code reusability in robotics. "The Robot Operating System 2 (ROS2)" by [3] is another noteworthy study. In this section, we summarized details of ROS developments and enhancements. It emphasizes performance, scalability, real-time capabilities, and expanded support for many programming languages and platforms. ROS2 intends to solve some of ROS1's drawbacks by providing a more robust and dependable platform for constructing complicated robotic systems.

Robot Operating System (ROS), an open-source middleware suite [2] widely adopted for robot software development. This chapter covers the goals of ROS, its design principles, the libraries and tools it provides, and details of ROS versions. ROS is not an actual operating system, it provides a collection of software frameworks for developing robot software. It offers various services for heterogeneous computer clusters, including hardware abstraction, low-level device control, implementation of commonly used functionalities, message-passing between processes, and package management. The execution of ROS-based processes is represented by a graph architecture, where nodes perform various tasks such as receiving, posting, and multiplexing messages related to control, state, planning, actuators, and sensor data. It is important to note that although ROS is not designed for real-time systems, it can be utilized in conjunction with real-time computing software to address low latency and responsiveness requirements for robot control.

2.1.1 Use of ROS

ROS primarily aims to facilitate code reuse in research and development. ROS offers distributed processes that allow for independent and loosely connected runtime execution. These processes can be categorized into simple stacks and packages to share and distribute. The ROS platform's design, from the filesystem to the community level, allows for independent development and implementation decisions. Further advantages of the ROS framework include its small architecture, support for all modern languages, including Python, C++, and Lisp, and its built-in unit, called ROS-test.

2.1.2 ROS Computation Graph Model

In the ROS framework, nodes are the individual processes that form a graph structure. These nodes communicate with each other via topics, which are connected by edges. Additionally, nodes can make service calls to other nodes, provide services to other nodes, and access shared data from the parameter server. The ROS master facilitates this communication by establishing node-to-node connections for topics and managing changes to the parameter server. Once nodes register themselves with the master, the master establishes direct peer-to-peer connections between the nodes, eliminating the need for messages and services to pass through the master.

2.1.3 Nodes

In the ROS Graph, a node represents a single process, and each node must register its unique name with the ROS master before carrying out any further operations. It is important to note that multiple nodes with different names can exist under different namespaces, and nodes can also be identified anonymously. Nodes serve as the central component in ROS programming, where client code typically takes the form of a ROS node. These nodes interact with other nodes by receiving information, taking corresponding actions, and sending information to other nodes.

2.1.4 Master

The ROS master plays a crucial role in facilitating the discovery of ROS nodes. It enables nodes to locate and establish communication with each other in a peer-to-peer manner. Once the nodes have successfully discovered one another through

the master, they can establish direct communication channels and exchange information seamlessly.

2.1.5 Topics

Topics in the ROS framework serve as a communication channel for nodes to exchange messages. Each topic has a unique and namespace-restricted name to ensure proper identification. The publish-subscribe model is employed, where nodes can publish messages to a specific topic and subscribe to receive messages from a topic. This anonymous approach allows nodes to transmit and receive messages without revealing their identities to other nodes. Only the content of the messages, which can include directives, state information, sensor data, control data, or other relevant information, is transmitted over the topic.

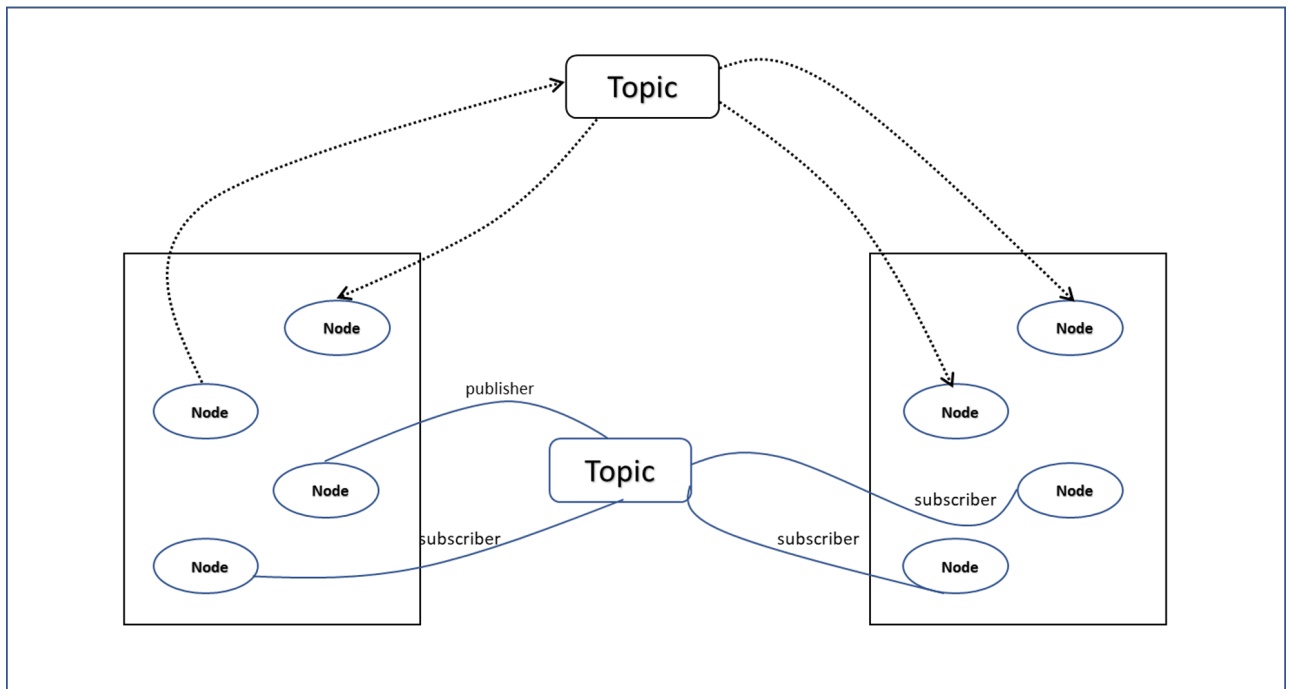


Figure 2.1: Illustrates ROS nodes publish and subscribe to topics.

2.1.6 Services

Nodes in the ROS system can provide services that define specific actions with a well-defined outcome. Services are commonly used for actions with a clear start and end, such as capturing a single frame or executing a hardware command. Nodes can advertise their services to other nodes and call services others provide, enabling interaction between them.

2.1.7 Parameter Server

The parameter server in the ROS system acts as a distributed database, offering public access to static and semi-static data. It is utilized to store data that remains relatively unchanged or maintains a constant value throughout the execution of the system. The parameter server is distributed among the nodes, allowing them to retrieve and utilize the stored data as needed.

2.2 Libraries and Tools

2.2.1 Gazebo Simulator

Gazebo Simulator is a powerful robotics research and development tool [4] connected with ROS. It provides a realistic physics-based simulation environment for testing robot behaviors and algorithms prior to deployment. The integration facilitates effective communication by allowing smooth data transmission between ROS nodes and the simulation. The visualization features of Gazebo allow for real-time debugging and validation of perception algorithms.

It also allows for the visualization of robot models and positions, which aids in comprehension and examination. The connection with ROS encourages compatibility with other ROS tools and libraries, facilitating the building of sophisticated robotic systems. Overall, when paired with ROS, Gazebo Simulator provides a complete solution for expediting development, optimizing algorithms, and enhancing the performance of robotic systems.

2.2.2 Point Cloud Library

The Point Cloud Library (PCL) is a widely used open-source framework [5] that revolutionized 3D perception and processing. PCL provides a complete collection of methods and tools for dealing with and modifying point cloud data, allowing researchers and developers to extract useful information from 3D sensor data.

Point clouds are dense collections of 3D points depicting objects and scenes' shapes and structures. PCL provides diverse point cloud processing techniques, including registration, segmentation, filtering, feature extraction, and surface reconstruction. Users may utilize these algorithms to analyze raw 3D sensor data, extract valuable characteristics, and segregate items of interest within a point cloud. PCL is suitable for various data sources since it supports several types of 3D sensors, including RGB-D cameras, LiDARs, and stereo cameras. The li-

brary offers tools for interactive data exploration and efficient data structures for handling and storing point clouds.

Additionally, PCL offers connections with other well-known libraries, including OpenCV and Eigen, to further expand its functionalities. Robotics, computer vision, augmented reality, and autonomous driving are industries where PCL is used. Using PCL, tasks like environment mapping, object detection, and robot localization are made possible in robotics. It allows 3D scene interpretation, reconstruction, and augmented reality in computer vision. In autonomous driving, PCL is essential in point cloud-based perception for object recognition, tracking, and scene interpretation. PCL is continuing to develop 3D perception, opening up ground-breaking applications across numerous industries.

2.2.3 OpenCV

OpenCV is a widely-used open-source library [6] for machine learning and computer vision algorithms. It offers a comprehensive collection of pre-implemented methods and functions for various computer vision applications. With over 2000 optimized algorithms, it encompasses a combination of classic and state-of-the-art techniques in computer vision and machine learning. The integration of ROS with OpenCV enables the utilization of powerful features such as object detection, tracking, and identification. Additionally, ROS extends OpenCV's capabilities through libraries like image pipelines, enabling functionalities such as camera calibration, monocular and stereo image processing, and depth image processing.

2.2.4 ROS Bag

ROSBAG is a robust tool in the Robot Operating System (ROS) ecosystem that enables users to capture, save, and replay ROS-targeted data. It functions as a flexible data logging and playback system, allowing researchers and developers to collect and analyze data from various sensors and robot components.

ROSBAG works by recording messages published on ROS topics over a set of periods. Any information communicated between ROS nodes can be included in these messages, such as sensor data, control instructions, and others. The captured data is kept in a message storage container and treated as a bag file.

One of ROSBAG's primary features is its ability to capture and playback data offline. This implies that the recorded bag file can be played back later, allowing users to examine and analyze the data without needing the original device or sensors. This capability is beneficial for debugging and testing since it allows

developers to analyze system behavior, assess algorithms, and detect problems in a controlled and repeatable manner.

Furthermore, ROSbag is not confined to recording and playing back on a single machine. Bag files are exchanged and moved across different ROS environments, allowing data sharing and cooperation across platforms. ROSbag extends ROS's capabilities by allowing data-driven research and assessment of robot systems

2.2.5 Rviz

RViz is a powerful visualization tool [7] frequently used in robotics to help with robot perception, planning, and debugging. It is a component of the Robot Operating System (ROS) architecture that provides a user-friendly interface for visualizing robot sensor data, robot models, and planning trajectories in 3D.

RViz allows users to view many sorts of sensor data, such as point clouds, laser scans, camera pictures, and odometry. This enables the visualization and study of the robot's perception of its surroundings in real-time. Robot operators and researchers may get valuable insights into the robot's behavior, uncover possible difficulties, and validate the efficiency of perception algorithms by visualizing sensor data. It's adaptability and extensibility make it an invaluable tool for both research and development.

RViz has a plethora of plugins and options that allow users to tailor the visualization to their requirements. Furthermore, RViz interfaces effortlessly with other ROS packages, allowing data sharing and interaction with other robot system components.

2.2.6 Catkin

The structure of ROS is built using the Catkin build system. Based on CMake, Catkin is a cross-platform, open-source, and language-independent build system used in ROS.

2.2.7 ROS bash

The utility to extend the capabilities of the bash shell is provided by the rosbash package. These tools, which replicate the functions of `ls`, `cd`, and `cp`, include `rosls`, `roscd`, and `roscp`. We used the ROS package name instead of the file path where the package is placed, thanks to recent updates to the `ros`.

2.2.8 ROS launch

The roslaunch utility enables the launching of multiple ROS nodes either locally or remotely, while the ROS parameter server allows for the specification of parameters. By utilizing XML-based configuration files, the roslaunch tool simplifies the process of automating complex startup and configuration tasks into a single command.

2.2.9 ROS Versions

Distribution Name	Release Date	Image
ROS Noetic Ninjemys	May 23 , 2021	
ROS Melodic Morenia	May 23, 2018	
ROS Lunar Loggerhead	May 23, 2017	
ROS Kinetic Kame	May 23, 2016	x 

Table 2.1: Depicted ROS versions published in numerous years.

2.3 Object Detection Model

Fast R-CNN, a popular deep learning architecture for object detection [8], has limitations such as computational intensity, difficulty in handling small objects, lack of temporal consistency, and the need for extensive labeled training data for each object class. To address these limitations and achieve precise boundary detection, researchers have extended Fast R-CNN with techniques like Mask R-CNN. Mask R-CNN, introduced by He *et. al.*, (2017), incorporates pixel-level instance segmentation, enabling accurate object masks and precise boundary detection in autonomous driving scenarios. This influential paper introduces Mask R-CNN [9], a state-of-the-art instance segmentation method. While this paper focuses on the strengths and advantages of Mask R-CNN, it indirectly highlights some limitations of instance segmentation. For example, it mentions the computational complexity associated with generating instance-specific masks and the challenges in handling overlapping instances.

An instance-aware semantic segmentation method [10] that combines semantic segmentation with instance-specific localization. Although it emphasizes the benefits of instance-awareness, the paper indirectly acknowledges the increased computational complexity and annotation burden associated with instance segmentation compared to semantic segmentation. Furthermore, this work proposes [11] a weakly supervised instance segmentation approach and highlights the limitations of instance segmentation. It discusses the challenges of handling occlusion and overlapping instances, as well as the higher annotation burden required for instance segmentation compared to semantic segmentation. The study aims to address these limitations by leveraging weak supervision to generate instance-level object masks. Semantic segmentation, a fundamental problem in computer vision, has advanced dramatically in recent years.

"Fully Convolutional Networks for Semantic Segmentation" was notable in this field [1]. The authors present a ground-breaking method for pixel-wise semantic segmentation that employs fully convolutional neural networks (FCNs). This work established the groundwork for future advances in deep learning-based semantic segmentation systems.

Another significant article is "DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs" by [12]. DeepLab, which integrates dilated convolutions and completely connected conditional random fields (CRFs), is proposed by the authors to improve segmentation accuracy. DeepLab's robust performance and real-time processing capabil-

ity have led to its widespread use in autonomous driving applications.

Furthermore, the study "ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation" by [13] introduces ENet, a lightweight network architecture intended exclusively for real-time semantic segmentation. ENet balances segmentation accuracy and computing efficiency, making it suited for resource-constrained applications like autonomous driving.

In addition, [14] paper "LinkNet: Exploiting Encoder Representations for Efficient Semantic Segmentation" proposes LinkNet, a network architecture that uses encoder representations to improve semantic segmentation efficiency. LinkNet achieves competitive performance using fewer processing resources, making it ideal for real-time autonomous driving applications.

Moreover, the paper "ICNet for Real-Time Semantic Segmentation on High-Resolution Images" by [15]. (2018 presents ICNet, a network architecture that addresses the challenge of real-time semantic segmentation on high-resolution images. ICNet utilizes a multi-scale and cascaded architecture to achieve accuracy and efficiency, making it suitable for high-resolution inputs often encountered in autonomous driving scenarios.

Next, We investigate the improvements and contributions of Neural Architecture Search (NAS) in the context of semantic segmentation for autonomous driving. Reinforcement learning combined with NAS has emerged as a powerful technique for automatically developing appropriate network designs that increase segmentation accuracy and efficiency in autonomous driving scenarios.

"Auto-DeepLab: Hierarchical Neural Architecture Search for Semantic Image Segmentation" by [16] is a seminal paper. To automatically determine the ideal architecture for semantic segmentation, the authors present a hierarchical search space and a reinforcement learning-based search method. On benchmark datasets, Auto-DeepLab achieves better performance and has been effectively applied to semantic segmentation in the context of autonomous driving.

The study "ENAS: Efficient Neural Architecture Search via Parameter Sharing" by [17] describes an effective NAS strategy that combines parameter sharing among child models. While the ideas of ENAS have not been primarily focused on semantic segmentation in autonomous driving, they have been adapted and applied to the task, resulting in breakthroughs in NAS for semantic segmentation.

The paper "Progressive Neural Architecture Search" by [18] offers a sequential model-based NAS optimization approach. This approach combines evolutionary search with reinforcement learning to identify designs with improved performance effectively. Progressive NAS has been used successfully for a variety

of tasks, including image classification and object recognition, and it has the potential to be extended to semantic segmentation in the context of autonomous driving.

We investigated several algorithms in our pursuit of performing the objective with minimal latency and improved accuracy. We started with the YOLO (You Only Look Once) architecture, explained in chapter 3, which we trained on the Berkeley dataset. This method produced positive results, displaying good accuracy across various settings. However, we ran across several limits that drove us to look into different strategies, eventually prompting us to investigate semantic segmentation.

Through thorough research, we extensively explored real-time semantic segmentation approaches to fulfill our objectives. We aim to discover an advanced deep-learning solution that would meet the specific requirements of our application. During our investigation, we identified a state-of-the-art method that exhibited exceptional performance. This approach underwent training using the well-established such as cityscape dataset, berkeley dataset and subsequently underwent evaluation using our own dataset. Chapter 3 covers an in-depth analysis and description of the datasets.

By employing this established technique, we aim to enrich the efficiency and precision of our autonomous driving system. Through meticulous experimentation and evaluation, we sought to address the challenges associated with real-time object detection and recognition, ultimately improving the overall performance and reliability of the system.

CHAPTER 3

YOLO Architecture

3.1 Introduction

Object detection, a fundamental problem in computer vision, involves locating and identifying objects within images or videos. This capability finds applications in diverse fields, such as robotics, autonomous driving, surveillance systems, and image understanding. In the context of our autonomous driving bot task, real-time and efficient object identification is crucial. We turn to YOLO (You Only Look Once), a renowned and influential framework to fulfill this requirement.

In this chapter, we explored the various improvements made in each version of YOLO, highlighting their evolution over time. Additionally, we provided a concise overview of the YOLOv5 architecture, which has gained significant traction. Furthermore, we presented the results of applying the YOLO model to three different datasets, assessing its performance and capabilities.

YOLO's real-time object detection abilities have attracted much interest in the computer vision world. By allowing objects to be recognized in one pass through the network, the initial YOLO architecture[19], announced in 2016, revolutionized the industry. However, later iterations of YOLO, such as YOLOv2 [20] and, YOLOv3 [21], YOLOv4 [22], addressed these issues and added enhancements to improve detection accuracy and handle objects of various sizes. The latest version, which is used, The YOLOv5 framework, distinguishes itself from other object identification frameworks with several distinctive features. It uses a simplified architecture that blends components from earlier YOLO iterations and integrates advance methods. With this architecture, YOLOv5 can reach desirable precision and real-time inference performance, making it appropriate for various applications.

The versatility and flexibility of YOLOv5 is one of its primary characteristics. It offers many model types in various sizes so that we can choose the best balance between model complexity, speed, and accuracy based on their unique re-

quirements. Additionally, YOLOv5 makes use of the PyTorch [23] deep learning framework, which makes it simple to develop, train, and deploy models.

3.2 Versions

Prior work of object detection uses the classifier to detect the object. Instead, YOLO formulates the issue of object detection as a regression to spatially separated bounding boxes and related class probabilities. In YOLO Architecture, a single neural network predicts the bounding box and its associated class probability in one pass.

3.2.1 YOLOv1

The initial iteration of YOLO [19] demonstrated superior speed compared to other object detection models. By treating detection as a regression problem, we eliminate the need for a complex pipeline. Our base network achieves a frame rate of 45 frames per second on a Titan X GPU, while a faster version operates at over 150 frames per second. This allows us to process streaming video in real-time with a latency of less than 25 milliseconds.

Furthermore, YOLO exhibits the advantage of leveraging the entirety of the input image during both the training and testing phases, thereby acquiring implicit knowledge about the contextual information and appearance of various object classes. This unique approach enables YOLO to acquire generalized representations of objects, ensuring robustness in its detection capabilities. When trained on real photos and assessed on artistic imagery, YOLO demonstrates superior performance compared to prominent object detection algorithms like DPM and R-CNN [24], surpassing them by a substantial margin.

Originally YOLOv1 was trained over the Pascal VOC benchmark dataset, which has fixed 20 classes. This model gives the flexibility to train over the custom dataset to predict the desirable classes according to the application. Despite its substantial contributions to the area, the first version of YOLO had numerous limits and disadvantages:

- **Localization Accuracy:** YOLO has trouble accurately localizing small objects. The model can have trouble accurately localizing items with fine details or those substantially smaller than the grid cell size since it divides the input image into a grid and assigns a bounding box to each grid cell.

- **Inability to Handle Overlapping Objects:** YOLO finds it challenging to locate and reliably detect things with close distances or overlaps. In some situations, the model might have trouble assigning the appropriate boundary boxes, which could result in overlapping or erroneous detections.
- **Limited Class Representation:** The initial YOLO model was built to find items in a predetermined set of 20 classes. It may only generalize well to items within the initial class set, even though it can be trained on custom datasets to detect more classes. This constraint limits the model's adaptability and applicability to a wider range of item detection applications.

3.2.2 YOLOv2

YOLOv2 (You Only Look Once version 2) [20] is an upgraded version of the original YOLO object detection framework. This release improves the performance of object detection by addressing a number of issues and adding new functionality. There are key improvements over the previous version of the YOLO :

- **Multi-Scale Training and Testing:** Multi-scale training and testing is one of YOLOv2's main improvements. The YOLOv1 model had a fixed input size, which restricted its capacity to identify objects of various sizes. By using images with different resolutions to train and test the model, YOLOv2 addresses this. Using this method, YOLOv2 can identify objects at various scales with higher precision.
- **Anchor Boxes:** Anchor boxes are a new idea introduced in YOLOv2 to enhance bounding box predictions. YOLOv2 predicts the offset values relative to a series of preconfigured anchor boxes of various sizes and aspect ratios rather than directly predicting bounding box coordinates. With the help of this technique, the model is better equipped to manage object variations, increasing the robustness and accuracy of localization.
- **Darknet-19 Architecture:** The network architecture used by YOLOv2 is known as Darknet-19 and comprises 19 convolutional layers. This architecture makes Better object detection performance possible, which enhances the model's total feature representation capabilities. Darknet-19 enables real-time object detection on various hardware platforms by striking a compromise between model complexity and processing performance.
- **High-Resolution Classifier:** A high-resolution classifier is incorporated into YOLOv2 to improve classification accuracy. This classifier is included at the

network's end and works with the final feature maps to provide more precise information for class predictions. The YOLOv1 grid structure's drawbacks are lessened by the high-resolution classifier, which also enhances the detection of small objects.

There are some disadvantages of this version of the YOLO, Even though this version has some improvements over the previous version:

- **Accuracy of Localization for Small Objects:** Like YOLOv1, YOLOv2 can have trouble correctly localizing small items, especially if they are considerably smaller than the anchor box sizes.
- **Difficulty in Handling Overlapping Objects:** YOLOv2 may have trouble accurately recognizing and localizing objects that are close to each other or overlap. In these circumstances, the model can have trouble designating distinct bounding boxes for various objects, resulting in overlapping or erroneous detections.
- **Higher Computational Requirements:** YOLOv2's enhanced architecture and multi-scale approach require more computational resources than the original YOLOv1.

3.2.3 YOLOv3

YOLOv3 (You Only Look Once version 3) [21] is a powerful object detection framework that builds on the achievements of its previous generations, YOLO and YOLOv2. In order to significantly improve the effectiveness of object detection, YOLOv3 fixes several issues from earlier versions and adds new features.

- **Improved detection accuracy of various scales:** One of the main goals of YOLOv3 is to improve detection accuracy. In order to accomplish this, YOLOv3 uses a feature pyramid network (FPN) architecture, which permits the extraction of multi-scale features. To better capture objects at various scales and improve detection performance, YOLOv3 makes use of features from various network layers with different spatial resolutions.
- **Darknet 53:** YOLOv3 also introduces the idea of "darknet-53," a more complex and effective network architecture than its predecessors. The darknet-53 backbone network uses advanced convolutional layers and residual connections to acquire detailed and abstract representations of objects, enhancing localization and discrimination.

Despite its improvements, YOLOv3 has certain drawbacks and restrictions. These consist of:

- **Higher computational demands:** The darknet-53 architecture's increased complexity and the multi-scale approach lead to higher computational demands than in previous versions.
- **Limited handling of overlapping objects:** YOLOv3, like its predecessors, may have difficulty accurately detecting and localizing closely located or overlapping objects.

3.2.4 YOLOv4

This version has come with a new architecture concept, giving better accuracy than the earlier version of Yolo [25]. Also, this version includes Some key features improved and added some new concepts:

- The primary objective of YOLOv4 [22] is to push the boundaries of object detection performance by leveraging the latest advancements in deep learning, network architecture design, and optimization techniques. It addresses several challenges encountered in previous versions and introduces innovative features to improve detection accuracy, handling of small objects, and overall efficiency.
- Implementing a more advanced backbone network called CSPDarknet53 is one of the main improvements in YOLOv4. This architecture makes greater information flow, feature reuse, and enhanced performance possible by combining the advantages of the Darknet-53 backbone from YOLOv3 with Cross-Stage Partial (CSP) connections. The CSPDarknet53 backbone improves the capabilities of feature representation, enabling more precise object detection.
- Many optimization approaches are also introduced by YOLOv4, such as the Mish activation function, which enhances gradient propagation and network convergence. It uses a modified spatial pyramid pooling (SPP) module that gathers features at multiple scales to help identify objects of various sizes. The PANet (Path Aggregation Network) module is also presented, aggregating features from several network stages to improve the capability to handle objects of different sizes.

Despite its advancements, YOLOv4 has limitations and disadvantages, including:

- With its innovative design and optimization methods, YOLOv4 requires a lot of processing power. In order to operate at peak efficiency, it could be necessary to have powerful hardware and enough memory.
- The performance of YOLOv4 is strongly influenced by carefully adjusting its hyperparameters, including learning rates, weight decay, and augmentation techniques. It frequently takes a lot of experimenting and parameter searching to get the finest results.
- The advanced design and optimization approaches of YOLOv4 cause longer training times relative to earlier versions. It can take longer to train YOLOv4 from the beginning, especially when working with massive datasets.

3.3 Architecture

Yolo's most recent version was used. Yolov5 architecture is made to give real-time object detection jobs an effective and adaptable solution. In order to increase detection accuracy and runtime performance, it integrates components from the previous version and introduces novel techniques. An overview of the YOLOv5 architecture is provided below:

- **Backbone Network:** A vital element of the YOLOv5 architecture, the backbone network is essential for extracting valuable characteristics from input images. It lays the foundation for further object detection phases and allows the model to record low-level and high-level visual data.

The CSPDarknet53 architecture is used as the foundation for the YOLOv5 backbone network. Cross Stage Partial connections, or CSP, is a method that improves the gradient flow during training and increases the network's capacity for learning. The Darknet backbone utilized in YOLOv4 was optimized for usage in the CSPDarknet53 backbone.

The backbone network comprises convolutions layers that gradually down-sample the input image. These layers use learnable filters to perform convolution operations to extract features at various spatial resolutions. Through the sequential application of these operations, the network gathers local and global contextual data, enabling the model to understand the input image.

The CSPDarknet53 [26] backbone uses many strategies to improve model effectiveness and feature extraction capabilities. It includes residual connections, which improve gradient propagation during training and ease the

vanishing gradient problem by facilitating the information flow through skip connections. These skip connections allow the network to concurrently learn fine-grained and high-level characteristics, capturing both semantic and detailed information.

In addition, the CSPDarknet53 backbone combines various convolutional layer types, including standard convolutions, dilated convolutions, and spatial pyramid pooling. The network is robust to objects of diverse sizes and aspect ratios thanks to these differences in convolutional procedures allowing the network to collect features at different receptive fields and scales.

YOLOv5 delivers a robust feature representation that aids in efficient object detection by utilizing the CSPDarknet53 backbone network.

- **Neck:** The neck acts as a link between the backbone network and the detecting head. Its foremost duty is to combine features gathered by the backbone network of varied resolutions and sizes and prepare them for object detection.

In YOLOv5, the neck comprises numerous layers, including a concatenation layer, a PANet layer, and several convolutions layers. The concatenation layer is in charge of merging backbone network features of various scales and resolutions, while the PANet layer is the duty of feature fusion.

PANet [27], which stands for "Path Aggregation Network", is a feature fusion module that increases network detection efficiency by merging features of various sizes. The PANet layer is divided into two parts: top-down and bottom-up pathways. The top-down pathway is in the role of upsampling and fusing features from higher-resolution feature maps, while the bottom-up pathway is for lower-resolution feature maps.

YOLOv5's neck comprises many convolutions layers that enhance the fused features, concatenation, and padding layers. These layers carry out operations like normalization, activation, and pooling to improve the expressiveness and discriminability of the feature representation.

- **Head:** The YOLOv5 architecture's last step is the head, responsible for generating bounding box predictions and class probabilities. It uses the fused features from the neck to perform the operations required to localize and classify objects in the input image.

In YOLOv5, the head comprises multiple prediction heads, each connected with a different stride. The stride specifies the spatial resolution of the fea-

ture map on which it operates. The heads work in parallel, allowing the model to identify things at different scales and locations at the same time.

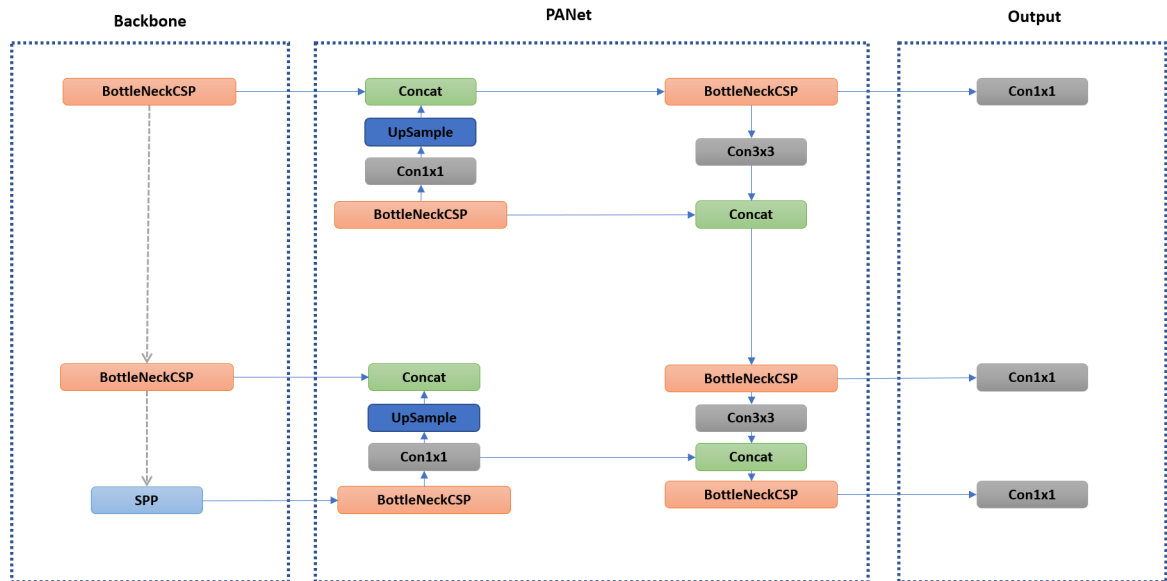


Figure 3.1: Illustrates an overview of YOLO architectures.

Each prediction head is comprised of convolutions layers followed by anchor boxes. The anchor boxes are predefined bounding box priors that capture the different aspect ratios and sizes of the training dataset's objects. The convolutions layers enhance the neck features and forecast bounding box coordinates and class probabilities.

Each head's prediction is then processed using non-maximum suppression (NMS) [28], eliminating duplicate bounding boxes and picking the most confident ones. Multiple predictions for the same object are merged into a single accurate bounding box by NMS, eliminating repeat detections.

The prediction heads in YOLOv5 utilize the benefits of an improved YOLO detection method. By combining anchor-based and anchor-free techniques, the model can handle objects of various sizes and aspect ratios better. By using this method, the detection accuracy is increased, and YOLOv5 is made robust to various object transformations.

The head in YOLOv5 also adds more methods to improve detection performance. Leaky ReLU (Rectified Linear Unit) and sigmoid activation functions introduce non-linearity and improve the modeling of complicated object boundaries and class probabilities. Additionally, methods like binary cross-entropy loss and focal loss are used to improve the model's parameters during training.

3.4 Dataset

The availability of high-quality datasets is critical for developing robust and efficient algorithms for autonomous driving. These datasets are essential for training and assessing computer vision models, which allow them to recognize and comprehend the complex real-world surroundings encountered on the road. Three notable datasets have emerged as invaluable resources in this context: our own dataset, the Berkeley DeepDrive BDD100K dataset, and the Cityscapes dataset. This chapter presents detailed technical information about all three datasets, accompanied by visual representations of the images. Furthermore, it includes a discussion on the dataset labeling tool

3.4.1 Berkeley Dataset

The Berkeley DeepDrive BDD100K dataset is a large-scale, diversified dataset intended for computer vision research and development, particularly on autonomous driving. It was developed by the Berkeley DeepDrive project in partnership with UC Berkeley researchers and the BDD Industry Consortium. BDD100K is an abbreviation for "Berkeley DeepDrive 100,000".

The BDD100K dataset includes various urban driving situations collected in various locations, weather conditions, and times of the day. It is made up of about 100,000 video sequences, each of which contains numerous frames. The dataset is annotated painstakingly with pixel-level semantic segmentation, item bounding boxes, lane markings, and other pertinent metadata. One of the BDD100K dataset's key characteristics is its extensive annotation schema. It provides detailed annotations for various autonomous driving-related objects, such as automobiles, pedestrians, bicycles, traffic signs, and traffic signals. These annotations allow researchers and developers to train and assess computer vision models in the context of autonomous driving for tasks such as object detection, semantic segmentation, and scene interpretation.

Furthermore, the BDD100K dataset is one of a kind in terms of diversity, including a wide range of driving scenarios and environmental conditions. Because of this variety, it is ideal for training and assessing robust models that can deal with real-world issues faced in autonomous driving applications. Figures 3.2 and 3.3 represent the images obtained from the BDD100K dataset.



Figure 3.2: Berkeley deep-drive dataset (1). Source: <https://bdd-data.berkeley.edu>



Figure 3.3: Berkeley deep-drive dataset (2). Source: <https://bdd-data.berkeley.edu>

3.4.2 Labelling Tool

LabelImg is a popular open-source graphical annotation tool for labeling images with bounding boxes for various computer vision tasks. It has been a popular alternative for annotating object detection datasets due to its user-friendly interface

and diverse capabilities. Users can use the tool to create bounding boxes around objects of interest inside a picture, which act as annotations to define the object's location and extent. It allows one to name several things within a single photograph, making it ideal for annotating complex scenarios.

Labellmg is remarkable for its support for numerous annotation formats, notably Pascal VOC and YOLO. This adaptability allows customers to select the format that best meets their specific requirements and integrate it easily with their preferred training pipeline or structure. Furthermore, the tool supports customizable class labels, allowing users to specify and apply labels to the annotated objects. We used this tool (figure 3.4) and labeled the 300 images from the Berkeley deep drive dataset in the format of YOLO.

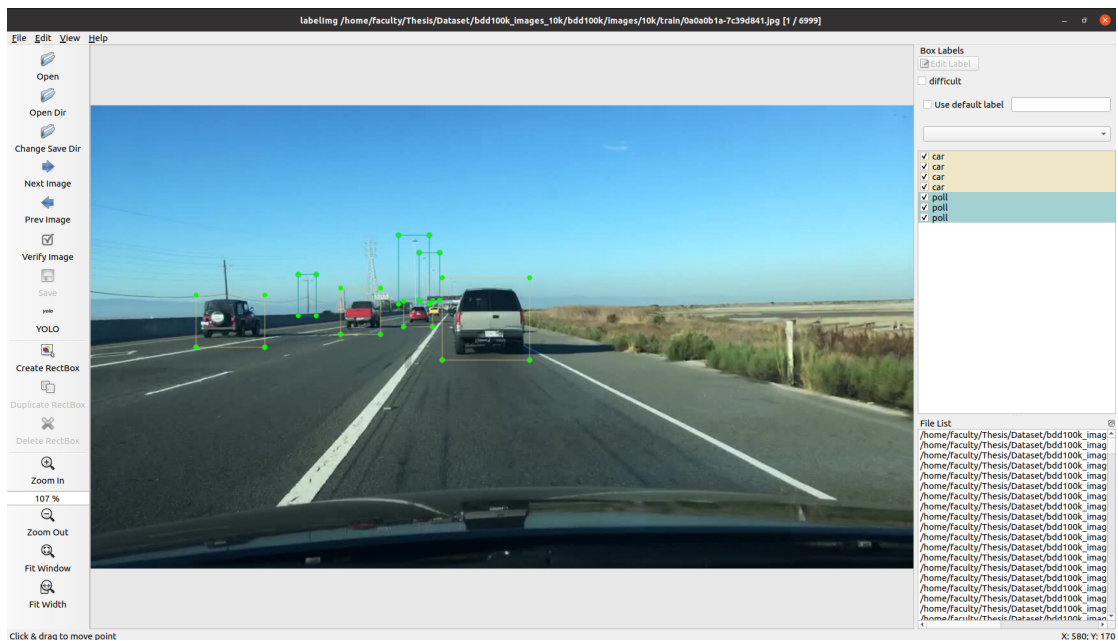


Figure 3.4: Labellmg GUI tool.

3.4.3 Cityscapes Dataset

The Cityscapes dataset is popular and influential in computer vision research [29] for autonomous driving. It is specifically developed to overcome the difficulties of interpreting and seeing urban street scenes. The dataset contains pixel-level annotations of high quality for a variety of semantic segmentation tasks.

There are two major components in the Cityscapes dataset: img8bit and gtFine. High-resolution RGB images recorded from stereo video sequences make up the img8bit component. These photographs depict real-world urban views [30] in various lighting, weather, and traffic scenarios. The photos are saved in 8-bit

PNG format, which provides a wealth of visual data for training and assessing computer vision models.

The Cityscapes dataset's gtFine component includes pixel-level annotations for semantic segmentation. It includes annotations for 19 classes, including roads, sidewalks, buildings, plants, cars, pedestrians, and various object and structural types. Each pixel in the image is labeled with a different class, allowing for a more detailed comprehension of the urban environment. These annotations are saved as PNG pictures, each pixel representing a different class label.

Combining the img8bit and gtFine components in the Cityscapes dataset enables researchers to train and evaluate semantic segmentation models for urban scene understanding. Using pixel-level annotations, Computer vision algorithms can effectively segment and identify distinct objects and regions inside urban environments. This dataset has aided in advancing algorithms and models for tasks including object detection, instance segmentation, and scene understanding in the context of autonomous driving. The Cityscapes dataset repository consists of original images (figure 3.5) stored in the "img8bit" directory, with the corresponding labeled images (figure 3.6) located in the "gtfine" repository.



Figure 3.5: Image from img8bit Cityscapes.

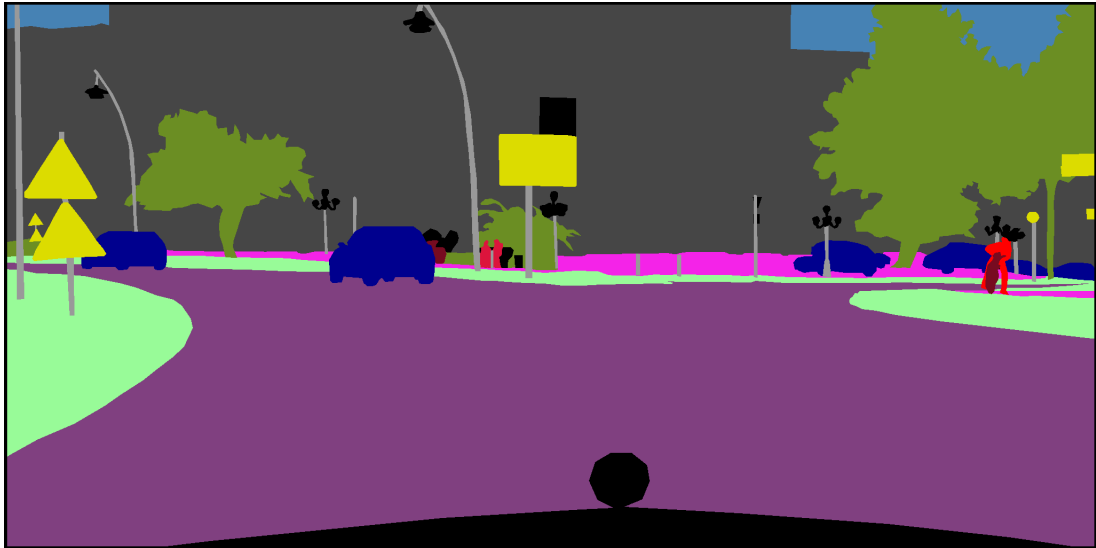


Figure 3.6: Image from gtFine Cityscapes.

3.4.4 Our Dataset

We have developed our own dataset specifically designed to evaluate the performance of used models in the context of the Indian scenario. While there are publicly available benchmark datasets, they may not encompass all the diverse and complex scenarios encountered in India, such as a mix of structured and unstructured data, rural and urban areas, varying road conditions, and diverse traffic scenarios. Hence, we created our dataset to address these specific requirements and evaluate the proposed segmentation model’s performance in an Indian context.

For data collection, we utilized a Samsung S23 model capable of capturing videos at a resolution of $1028 * 2048$ pixels. The videos were recorded at a consistent frame rate of 60 frames per second. In total, our dataset comprises five videos, each with a duration of 15 minutes. By capturing real-world footage using this setup, we aimed to simulate the Indian driving environment and provide a comprehensive dataset for evaluation purposes. we present two images from this repository. These images offer insights into the unique characteristics and challenges of driving in India.



Figure 3.7: Image from our dataset (1).



Figure 3.8: Image from our dataset (2).

3.5 Verification of YOLO Model

3.5.1 Mode Variants

Yolo version 5 architecture offers multiple models variant. Each model has a different size and complexity represented in Table 3.1. We used the miniature model, which better suits our application and gives better results with minimum latency.

Model	mAP	Parameters	Flops
YOLOv5n	28.0	1.9	4.5
YOLOv5s	37.4	7.2	16.5
YOLOV5m	45.4	21.2	49.0
YOLOV5l	49.0	46.5	109.1
YOLOV5x	50.7	86.7	205.7

Table 3.1: Overview of YOLOv5 different model.

3.5.2 Training

The YOLOv5s model includes pre-trained weights trained on large-scale datasets such as COCO (Common Objects in Context) [31] to capture generic object recognition features. These pre-trained weights can fine-tune the model on the target dataset or directly infer new photos.

Employing the pre-trained YOLOv5s model can benefit from the knowledge and representations obtained during the pretraining phase. These pre-trained weights give the model’s parameters a good start, allowing it to converge faster during fine-tuning and achieve more remarkable performance. The trained YOLOv5s model captures a wide range of objects and their visual features from the COCO dataset, which has 80 object categories. It learns to detect things of diverse sizes, aspect ratios, and positions, allowing it to generalize effectively to other recognition of objects tasks.

We used this pre-trained weight of yolov5s and trained over the benchmarked dataset, Berkeley, and labeled this dataset with 14 different classes according to our application needs. We labeled 300 images and divided them into training and validation. We labeled only 300 images with 14 classes to experiment on the YOLO model. These images are divided into 60% (180 images) training, 30% (90 images) validation, and 10% (30 images) testing.

The hyper parameters were carefully configured during the training phase based on the available resources. All relevant information regarding the hyper parameters is presented in the provided Table 3.2.

It is essential to evaluate the model’s performance on a validation set during training to assess its accuracy and make necessary improvements. Mean average precision (mAP) and other evaluation measures are frequently used to assess the model’s detection performance. This model uses the mAP to measure the performance. Additionally, two different matrices are used to measure the performance, mAP@50 and mAP@50-95. The mAP@50 is the mean average precision at a 50% detection threshold. It computes the model’s precision and recall when identified detections with an intersection over union (IoU) overlap of 50% or greater with the

Parameters	Value
Input Image Size	1024 x 1024 x 3
Batch Size	16
Learning Rate	0.01
Optimizer	Stochastic Gradient Descent
Epochs	100
Momentum	0.937
Weight Decay	0.0005

Table 3.2: Hyper-Parameters

ground truth bounding boxes. It assesses the model’s ability to recognize items accurately within a moderate overlap range with the ground truth. The mAP@50-95 broadens the evaluation range by considering a broader range of IoU thresholds ranging from 50% to 95%. It computes the average precision over these thresholds, thoroughly evaluating the model’s accuracy and precision across varying levels of overlap with the ground truth. mAP@50 and mAP@50-95 are useful indicators for assessing the performance of object identification models. They shed insight into the model’s capacity to detect objects properly and consistently when varied levels of IoU overlap are considered. Higher mAP values suggest better performance and object-detecting capabilities.

During the training phase, the YOLOv5 model undergoes evaluation on the validation dataset. The performance of the validation dataset is represented in a Table 3.3 that includes the count of images and instances for specific classes. Additionally, the table displays the precision and recall metrics and the mean average precision (mAP) value.

Class	Images	Instances	P	R	mAP50	mAP50-95
Car	33	142	0.234	0.972	0.806	0.0496
Poll	33	40	0.22	0.35	0.194	0.0656
Tree	33	19	0.0914	0.158	0.102	0.0372
House	33	7	1	0	0	0
Sign Board	33	21	0	0	0.0154	0.00677
Traffic Light	33	13	0.265	0.385	0.222	0.076
Person	33	17	0.345	0.125	0.213	0.0544
Truck	33	7	1	0	0.0467	0.0156
Divider	33	9	1	0	0	0
Bicycle	33	4	1	0	0	0
Bus	33	4	1	0	0.00485	0.000485
Pedestrian Crossing	33	1	1	0	0	0

Table 3.3: Model performance summary on validation dataset.

3.5.3 Results

The model's performance is represented using matrices that measure its effectiveness. Specifically, we evaluated the model's performance during the training process using metrics such as the F-score and precision-recall matrix. These matrices are valuable tools for assessing the model's performance and understanding its accuracy and effectiveness. In the following discussion, we concisely describe each matrix and present the corresponding results, offering insights into the model's performance evaluation.

The confusion matrix is a popular method for assessing the performance of a classification model, including object detection models such as YOLO. It provides a comprehensive view of the model's predictions and the actual ground truth labels, allowing various assessment metrics to be calculated. The confusion matrix presents in figure 3.9 the performance evaluation of our model across 14 different classes. The x-axis of the matrix corresponds to the predicted values generated by the classifier, while the y-axis represents the true values. This matrix provides a comprehensive overview of the model's performance regarding correctly and incorrectly classified instances for each class.

Further, Precision and recall are two evaluation measures frequently employed in machine learning and information retrieval applications, such as object detection. Precision measures the model's ability to correctly identify positive instances among the positive examples it predicts. It calculates the fraction of true positive predictions made by the model out of all positive predictions. In other words, precision tells how many of the positive predictions are right. Precision is calculated as $TP / (TP + FP)$, where TP is the number of correct predictions, and FP is the number of incorrect guesses. A high precision number suggests that the model makes accurate positive predictions and has a low rate of false positives.

The model's capacity to detect every positive instance in the dataset is measured by a recall, also known as sensitivity or true positive rate. It calculates the fraction of true positive predictions among all positive instances. Recall reflects how many positive instances the model accurately captures. The recall is calculated as $TP / (TP + FN)$, where TP is the number of correct predictions, and FN is the number of incorrect guesses. A high recall number implies that the model has a low false negative rate and can detect most of the positive cases in the dataset.

Precision and recall have an inverse relationship. As the decision threshold for forecasting positive instances is raised, the precision rises while the recall falls. Lowering the choice threshold, however, boosts recall but may reduce precision. The precision-recall trade-off is essential when determining the ideal balance be-

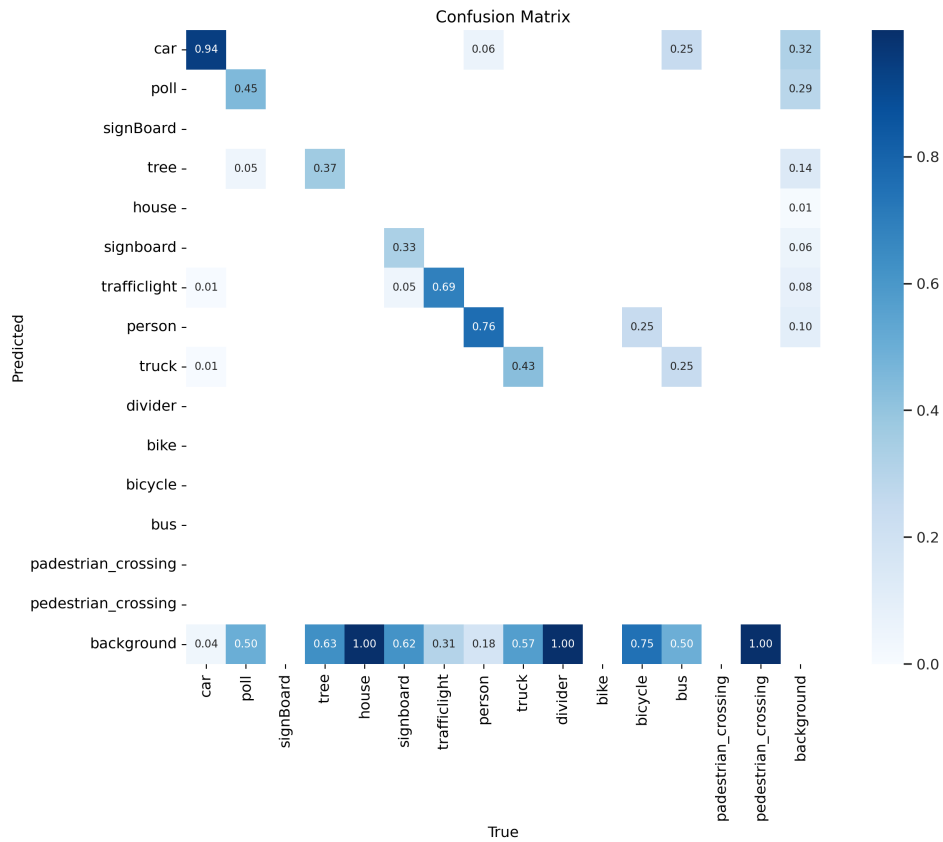


Figure 3.9: Illustrates the confusion matrix.

tween the two metrics. A higher value of the precision-recall curve indicates better performance in terms of precision and recall. Inferring more precise positive predictions and effectively identifying a higher percentage of positive cases predicts a reduced rate of false positives and false negatives.

The performance of our model is visualized in figure 3.10 through the Precision-Recall curve . The figure illustrates that the model tends to make positive predictions for multiple classes, while the values for other classes are close to zero. Our observations indicate that this behavior is influenced by the training dataset, which contains a limited number of instances for those particular classes. The count of instances for each class is provided in the accompanying Table 3.3, highlighting the potential reason behind the model’s suboptimal learning for those specific classes.

The F1 curve provides insight into how precision and recall trade-off at certain decision thresholds. It combines accuracy and recall into a single metric that considers the model’s ability to detect positive occurrences properly and minimize false positives. It is a balanced assessment of the model’s overall performance because it is the harmonic mean of precision and recall. Out Yolo Model predictions

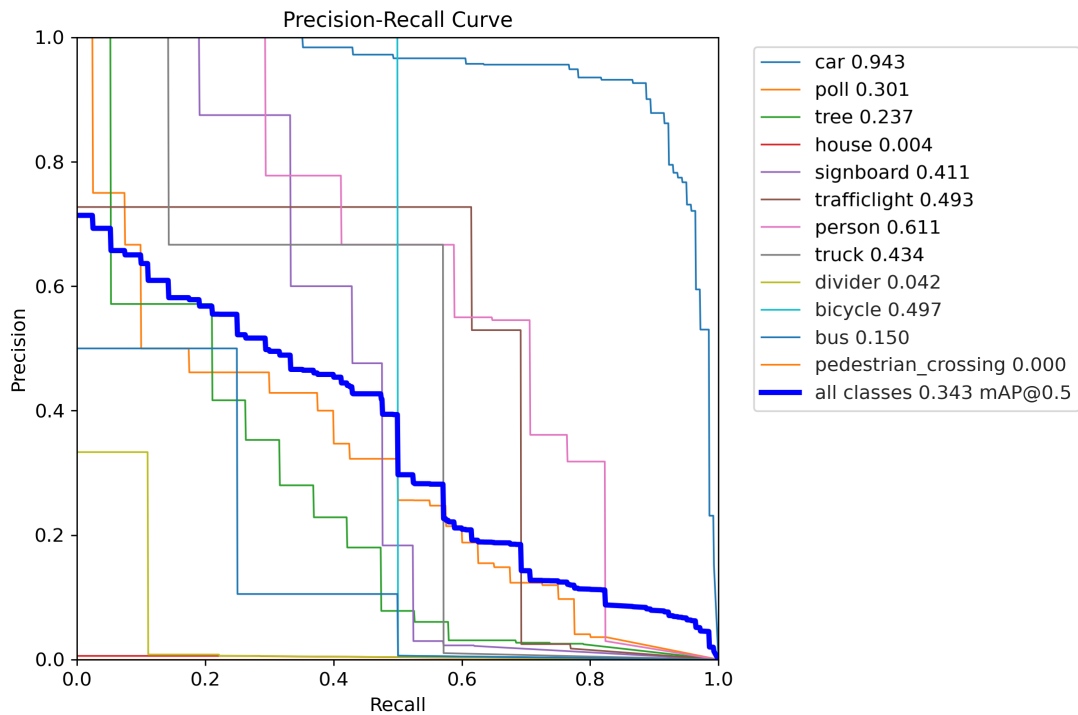


Figure 3.10: Precision-recall curve.

are tested at multiple decision thresholds to generate an F1 curve. The decision threshold specifies the level of certainty at which a predicted bounding box is considered a positive detection. Different trade-offs between precision and recall can be detected by adjusting the decision threshold. The F1 curve depicts the F1 score at various decision thresholds, visually depicting the model's performance over various operational points. In the experimental of our application, the F1-score of all classes was 0.32 at the threshold 0.093, shown in figure 3.11

Our YOLO model underwent testing on diverse data sets to evaluate its performance across various environments. Three distinct scenarios are considered for this purpose. The first scenario involved a low-traffic environment, the second scenario featured a moderate level of traffic, and the third scenario simulated peak traffic conditions on the road.

The image in figure 3.12 from the cityscapes dataset depicts objects situated at considerable distances from the car's camera and captured under low-light conditions. Consequently, the model's performance (see figure 3.13) in predicting certain classes, such as trees, poles, and traffic lights, was adversely affected. This can be attributed to the inherent difficulty in accurately detecting and classifying small-sized objects, particularly under such challenging conditions.

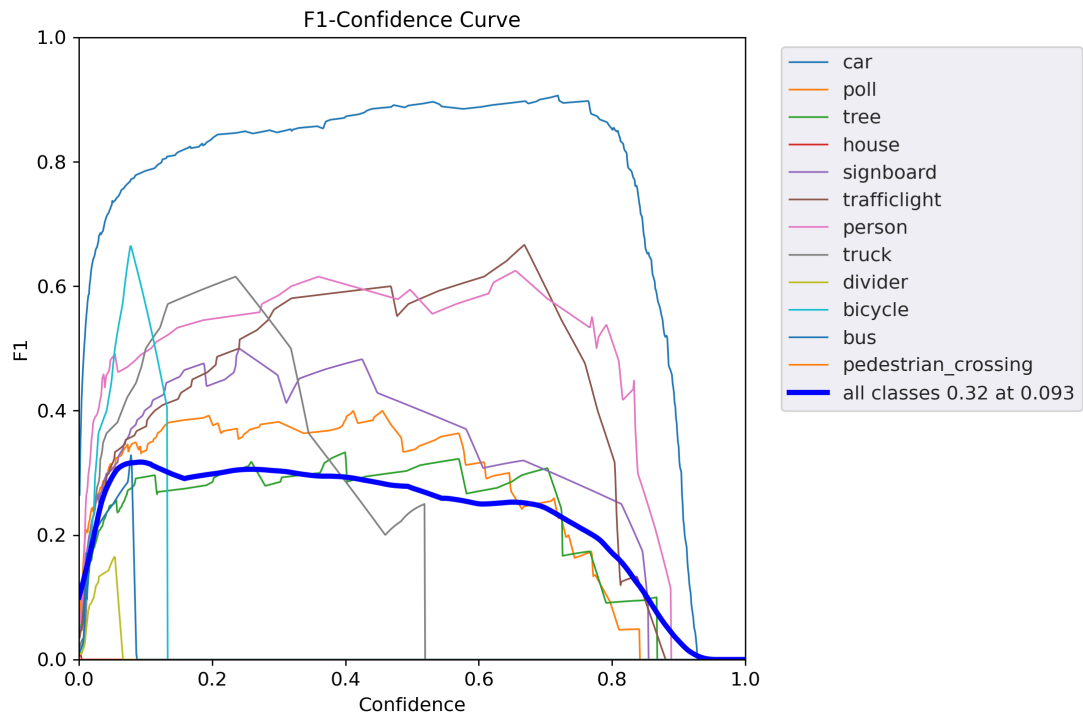


Figure 3.11: F1-score curve.



Figure 3.12: Original Cityscape dataset image.

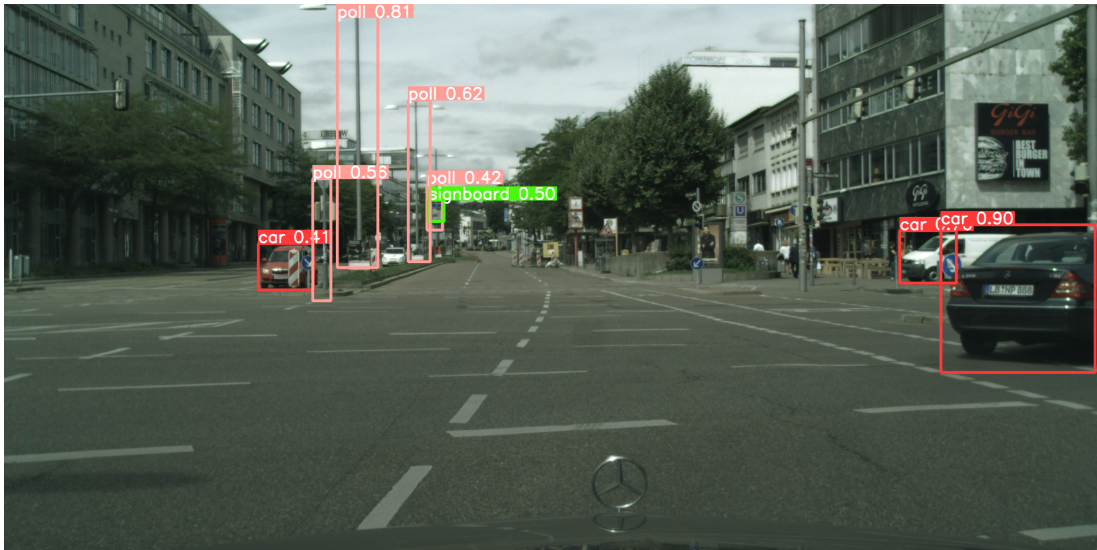


Figure 3.13: Resultant image of Cityscape dataset.

Another image (see figure 3.14) from the Berkeley dataset exhibits shadows on certain parts of the road, with limited areas illuminated by light. Under these conditions, the YOLO model performs better than the cityscapes dataset as depicted in figure 3.15. It successfully predicts all classes within the image, showcasing its enhanced object detection and classification capability.



Figure 3.14: Resultant image of Cityscape dataset.

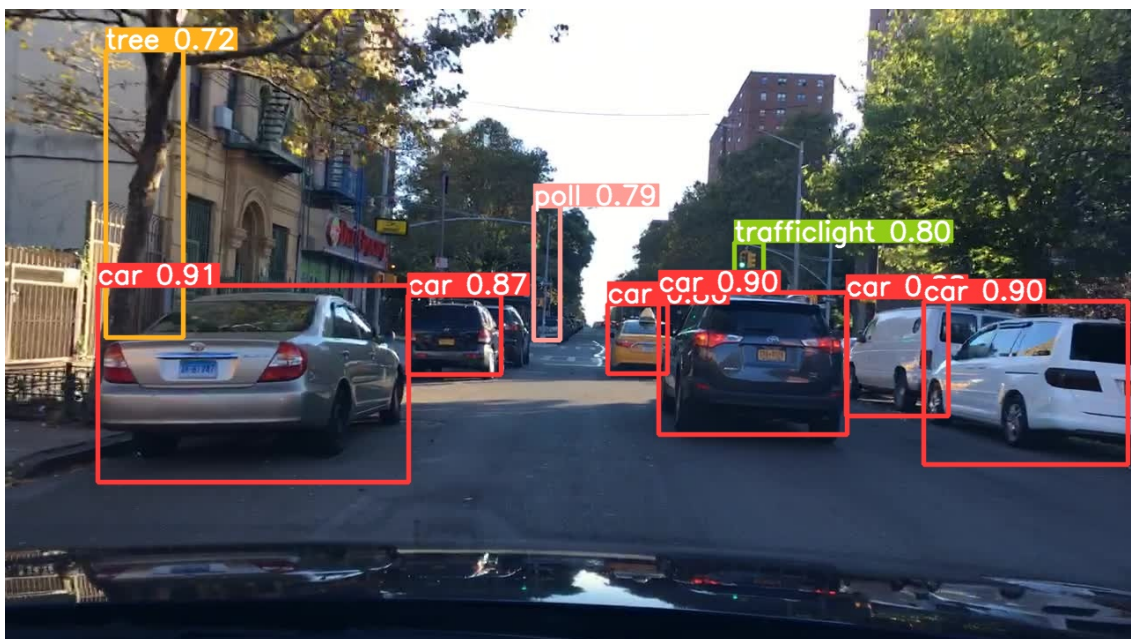


Figure 3.15: Resultant image of Cityscape dataset.

Further, the image (see figure 3.16) from our custom dataset portrays complete roads with vehicles and the presence of trees and poles. Notably, the model performs (see figure 3.17) well in this scenario, successfully detecting all the classes within the images. However, it occasionally misclassifies certain instances, such as mistaking an auto-rickshaw or a small truck for a car. Despite these occasional mispredictions, the model's overall performance remains commendable in accurately identifying and labeling most classes in the images.



Figure 3.16: Resultant image of Cityscape dataset.

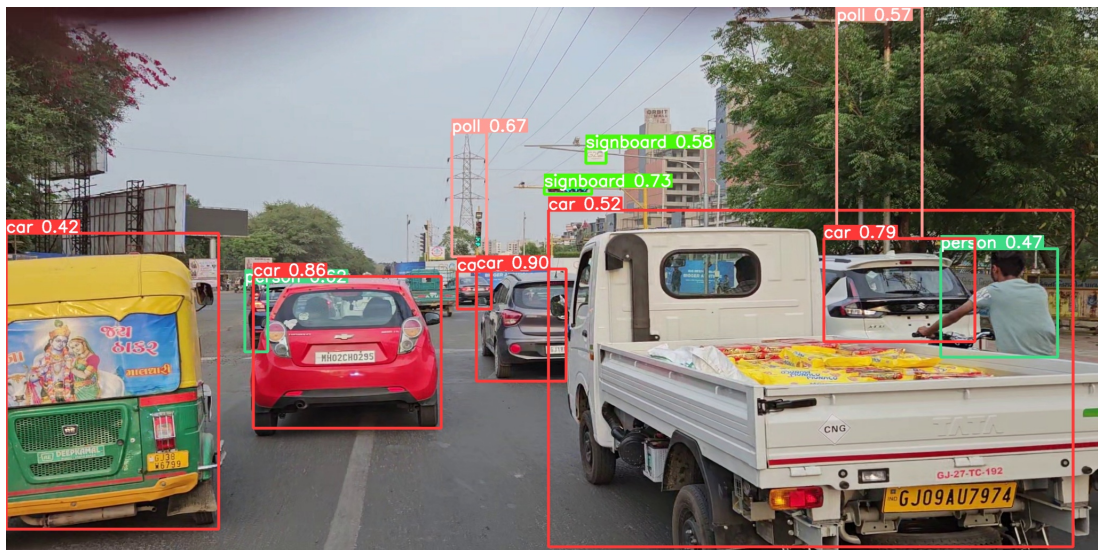


Figure 3.17: Resultant image of Cityscape dataset.

In conclusion, the YOLO model demonstrates good performance across various environmental conditions. However, further improvement can be achieved

by training the model on a larger dataset encompassing a more comprehensive range of light conditions and smaller objects. Exposing the model to a more diverse training set is expected to exhibit enhanced performance, surpassing the current capabilities of the model.

3.6 Limitation

Despite the YOLO architecture’s reputation for robustness and exceptional performance in object detection, it poses certain limitations for our research system. One of these limitations is the lack of precise boundary information provided by its bounding box output, which is crucial for accurate object identification. Additionally, YOLO’s reliance on a single-scale feature map may impede its ability to effectively detect objects of varying scales, posing challenges in complex real-world driving scenarios.

To overcome these limitations, we explored alternative methods and opted for semantic segmentation, described in Chapter 4. By leveraging semantic segmentation, we obtain precise object boundaries, addressing the need for accurate object identification and recognition in our autonomous driving system. This decision allows us to enhance the reliability and effectiveness of our system, as semantic segmentation provides detailed and accurate object boundary delineation, surpassing the limitations of YOLO’s bounding box output.

CHAPTER 4

Semantic Segmentation

Semantic segmentation, which assigns a unique label to each pixel in an image, is critical in computer vision applications such as autonomous driving, scene interpretation, and robotics. Real-time semantic segmentation is critical in applications that require quick and precise perception. However, due to the computational complexity of existing approaches, attaining real-time semantic segmentation on resource-constrained devices remains difficult.

We used a novel approach dubbed FasterSeg to handle the difficulty of attaining real-time semantic segmentation [32]. Our goal is to improve the efficiency of semantic segmentation networks while maintaining a balance of speed and accuracy, especially on devices with low processing capabilities. Using neural architecture search techniques, FasterSeg seeks to construct optimized network architectures that can give rapid and high-quality segmentation results. We will delve into the intricacies of this technique in subsequent sections, including a system overview and a comprehensive discussion of the results obtained on various data sets to evaluate its performance. The primary motivation for FasterSeg derives from the fact that classic semantic segmentation networks, such as DeepLab, FCN, and PSPNet, frequently suffer from high processing demands, making them unsuitable for use in real-time applications [33]. To overcome this issue, we use a search algorithm to search a wide design space of network architectures for efficient building blocks and network configurations that maximize inference performance while maintaining segmentation accuracy.

Furthermore, FasterSeg employs the teacher-student distillation technique to deploy the model in lightweight devices. It transfers knowledge from an accurate, computationally expensive teacher network to a quick and effective student network. By balancing speed and precision, this information transfer enables the student network to perform segmentation in a competitive manner in real-world situations.

The search strategy of FasterSeg is guided by a comprehensive set of perfor-

mance metrics encompassing speed and accuracy evaluations. We utilize established metrics like mean Intersection over Union (mIoU) and frames per second (FPS) to evaluate the segmentation quality and inference speed. To explore the architectural space efficiently, we design a multi-objective optimization problem to strike the optimal balance between speed and accuracy. This enables the search algorithm to navigate through the possibilities effectively.

In order to validate the effectiveness of FasterSeg, extensive tests are conducted on various benchmark datasets, including Cityscapes, Berkeley deep drive, and our own dataset. We establish its superiority by comparing FasterSeg’s performance to state-of-the-art semantic segmentation methods, such as manually created networks and existing architecture search approaches and also showed the comparison with other models in section 4.6.2.

FasterSeg exhibits impressive enhancements in real-time performance, as evidenced by experimental results. The model’s training on a combination of the Cityscapes and Berkeley benchmarked datasets contributes to this notable improvement. Unlike the Cityscapes dataset, which focuses solely on urban environments, the Berkeley dataset encompasses a broader range of environments. This diverse training approach enables FasterSeg to achieve higher frames per second (FPS) while maintaining a competitive level of segmentation precision. These results underscore the effectiveness of FasterSeg in effectively balancing the trade-off between speed and accuracy.

4.1 Neural Architecture Search

Neural Architecture Search (NAS) is an automated method [33] for discovering optimal neural network architectures. This technique generates and evaluates alternative network architectures using a controller neural network commonly based on recurrent neural networks (RNNs). Based on the performance of the produced architectures, the controller network is trained using reinforcement learning to maximize a reward signal. The overview of the NAS is represented in figure 4.1. The Neural Architecture Search (NAS) process involves an iterative approach of sampling and evaluating network architectures based on decisions made by the controller network. These decisions encompass various aspects, such as the number and types of layers and their connections. The sampled architectures are then trained and evaluated to obtain performance metrics that guide the search for improved architectures. This iterative process allows for exploring different architectural configurations, leading to the discovery of architectures that demon-

strate better performance.

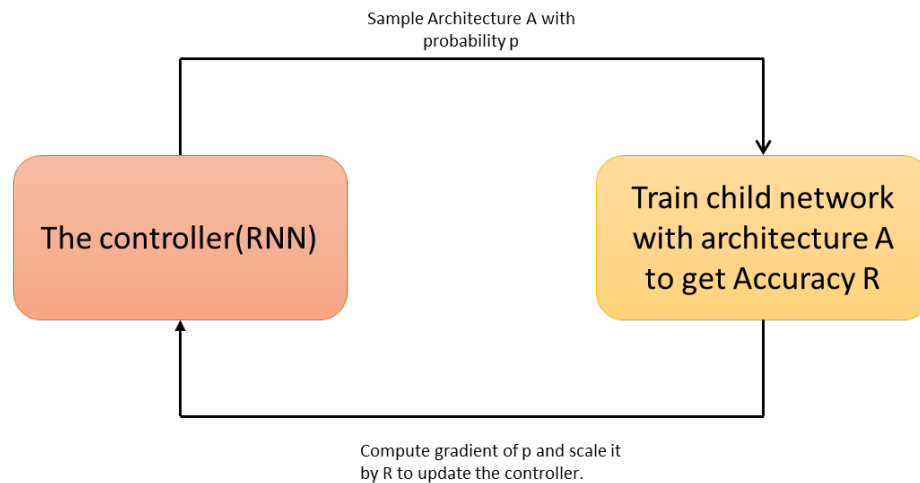


Figure 4.1: An overview of neural architecture search

4.2 Teacher/Student Co-searching For Knowledge Distillation

Teacher-student knowledge distillation is a machine learning technique to transfer knowledge from a more complex model (the teacher) to a smaller and more efficient model (the student). This approach [34] involves the mentorship of the student model by the teacher model during the training process, providing additional information.

The teacher model guides the student model by sharing its knowledge in the form of soft targets, which are the output probabilities of the teacher model or feature embeddings from intermediate layers. The primary objective of knowledge distillation is to enable the student model to acquire the generalization abilities and knowledge of the teacher model.

By leveraging the knowledge distillation technique, the student model can achieve comparable or superior performance to the teacher model while maintaining computational efficiency. This approach allows for the effective transfer of knowledge, enabling smaller models to benefit from the expertise and capabilities of larger models.

4.3 FasterSeg

Our FasterSeg architecture was developed based on a multi-resolution search space that draws inspiration from previous manual design achievements. To mitigate the "architecture collapse" issue, we incorporated fine-grained latency regularization techniques proposed by [35] in their InstaNAS work. Additionally, we leveraged a teacher-student co-searching framework to enhance our FasterSeg model further, resulting in a compact yet highly accurate student network.

4.4 Optimizing Search Space for Efficient Multi-Resolution Branching

FasterSeg incorporates a multi-resolution branching search space, allowing for optimization of branches with different output resolutions. The head module progressively aggregates these outputs. Each cell is independently searched, with variable downsampling speeds (s) and two inputs/outputs. The search process explores expansion ratios within a single super kernel. Efficient cells with searchable super kernels are automatically selected and combined to form branches of various resolutions.

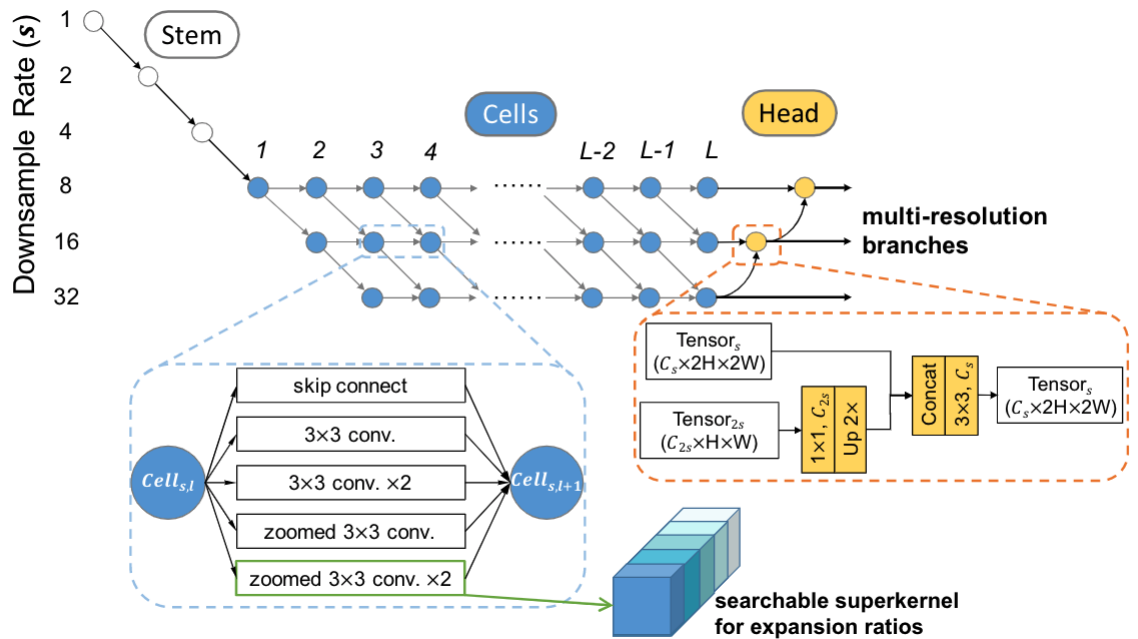


Figure 4.2: Overview of multibranch searching [32].
courtesy of <https://arxiv.org/abs/1912.10917>

4.4.1 Searchable Multi-Resolution Branches

Our framework introduces a novel approach by incorporating a multi-resolution strategy within L-layer cells (see figure 4.2). These cells are designed to receive input from two interconnected predecessors and generate two distinct feature maps with varying resolutions. This concept of leveraging multiple branches with different resolutions has demonstrated its efficacy in hand-crafted networks tailored explicitly for real-time segmentation applications [15] [36]. However, the current state of NAS algorithms has been constrained to exploring single backbone architectures, limiting their optimization potential.

We aim to select b branches with different final output resolutions, enabling investigation and decoding of each branch using backtrace. This approach allows our NAS framework to explore b individual branches gradually learned and aggregated by the head module (see figure 4.2).

To increase model capacity while minimizing latency, we adopt the tradition of increasing the number of channels with each resolution downsampling. We utilize our stem module to downsample the input image to $1/8$ of the original scale and set searchable downsample rates at 8, 16, and 32.

4.4.2 Selecting Optimal Operators for Enhanced Receptive Field Coverage

To optimize inference latency in our approach, we prioritize the execution speed of operators as a direct metric rather than relying on indirect metrics such as FLOPs (floating-point operations). One operator candidate that effectively reduces FLOPs and parameter count [37] is group convolution, as it achieves the same receptive field as standard convolution while being significantly faster. In contrast, dilated convolution [38], despite its comparable FLOPs and parameter count, exhibits higher latency. To address this discrepancy, we introduce a novel variation called zoomed convolution, which applies sequential bilinear downsampling, standard convolution, and bilinear upsampling to the input feature map. This design reduces latency by 40% compared to standard convolution and increases the receptive field by 2x. Within our search space, we include optimized operators that incorporate these advancements.

- skip connection
- 3 x 3 conv
- 3 x 3 conv. x 2

- Zoomed Conv : bilinear downsampling + 3×3 conv. ++ bilinear sampling
- Zoomed Conv. $\times 2$: bilinear downsampling + 3×3 conv. $\times 2$ + bilinear sampling

4.4.3 Searchable Super kernel For Expansion Ratios

To provide diversity to the choices of channel expansion ratios, we give each cell the ability to select different ratios. However, calculating the ideal connection width between succeeding cells poses a significant task due to the huge number of conceivable combinations. To overcome this, we present a differentially searchable super kernel implementation that directly examines and probes the expansion ratio within a single convolutional kernel. This super kernel supports a wide range of ratios designated as $U \subset N+$.

We use a strategic method during the architectural exploration phase in which a singular expansion ratio [39][40] is sampled, activated, and then back-propagated for each super kernel at each stochastic gradient descent step. Using the well-praised "Gumbel-Softmax" technique, we may streamline the overall supernet-work architecture while improving memory efficiency.

Within our search space, we stick to the standard practice of gradually increasing the number of channels as the resolution decreases. This is accomplished by defining the width as $U \times v$, where v has the values 8, 16, and 32.

4.5 Regularized Latency Optimization with Finer Granularity

Achieving reduced latency while keeping optimal performance is a difficult task. A previous study ([35] [41]) has discovered a reoccurring problem in the search process: the supernet or search strategy frequently becomes stuck in unfavorable "local minimums". These local minimums produce structures with reduced latency but impaired accuracy, especially in the initial phases of exploration. It has been discovered that the overuse of skip connections, rather than emphasizing low expansion ratios [42], contributes to the condition known as "architecture collapse". We present a novel technique that employs fine-grained and decoupled latency regularisation to address this issue.

Our analysis demonstrates that the supernet's susceptibility to varied operators (P), downsampling rates (d), and expansion ratios (U) is the primary cause of

the "architecture collapse" problem. Notably, operators such as 3×3 conv. 2 and zoomed conv have significant latency differences. Similarly, slender and broad expansion ratios show a noticeable delay difference. Downsampling rates such as "8" and "32" have insignificant differences because they involve doubling the input and output channels.

Building on these findings, we provide a systematic strategy to optimize latency through regularisation, using the various granularities covered by our search space. We measure supernet latency at three granularities (P, d, U) and use different regularisation factors for each facet.

$$Latency(P, d, U) = w_1 Latency(P|d, U) + w_2 Latency(d|P, U) + w_3 Latency(P|U, d)$$

4.6 Verifications

4.6.1 Architecture Search

The supernet architecture comprises a total of $H = 16$ layers, while the chosen downsample rate is denoted as $d = \{8, 16, 32\}$. In our investigation, the default setting for branches is $b = 2$, as introducing additional branches would lead to a substantial increase in latency. For each downsample rate d and layer H , we thoroughly explore a range of expansion ratios represented by $\chi_{d,H} \in U = \{4, 6, 8, 10, 12\}$. The multi-resolution branches encompass a comprehensive set of 1695 distinct pathways. Taking into account the combinations of cells and expansion ratios, the resulting search space comprises approximately $(1 + 4 \times 5)^{(15+14+13)} + 5^3 \approx 3.4 \times 10^{55}$ possibilities. This expanded search space presents a significantly larger and more formidable exploration challenge compared to initial investigations.

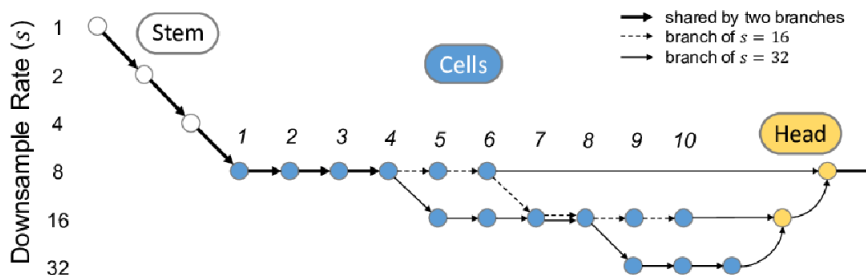


Figure 4.3: FasterSeg network discovered by our NAS framework [32].
courtesy of :<https://arxiv.org/abs/1912.10917>

The architecture search is carried out on the Cityscapes dataset. The image resolution used for this is $1024 * 2048$. 4.4 depicts the best spatial resolution found

in the FasterSeg Method. Also, our method can achieve multi-resolutions with appropriate depths. The first three operators are shared by both branches, after which they diverge and opt to aggregate outputs with downsampling rates of 16 and 32.

4.6.2 Exploring the Effectiveness of Multi-Resolution Search Space and Collaborative Search

To evaluate the efficiency of our NAS design, we do experiments on the Cityscapes dataset and Berkeley dataset. We look specifically at the effects of operators (P), distillation on correctness, expansion ratios (U), latency, and downsample rate (d). There is a trade-off between frames per second (FPS) and mean Intersection over Union (mIoU) as we shift from a single backbone ($b = 1$) to many branches ($b = 2$), illustrating the usefulness of the multi-resolution design for segmentation tasks. By experimenting with different expansion ratios (U), we uncover a quicker network with an FPS of 160 and a high accuracy of 70 %. This demonstrates the benefit of our searchable super kernel in minimizing duplicate channels while maintaining accuracy. Our collaborative research framework promotes the growth and advancement of the student network (S).

Congfiguration	MIoU%	FPS	FLOPs	Parameters
$P, d U = 8, b = 1$	66	177	27.0G	6.3M
$P, d U = 8, b = 2$	69.5	119.9	42.0G	10.8M
Teacher(T) and Student(S) Co-searching				
$S : P, d, U b = 2$	70	160	28.2G	4.4M
$T \rightarrow prunedT$	66.1	146.7	29.5G	4.7M
$T \rightarrow S$	73.1	163.9	28.2G	4.4M

Table 4.1: Studies of numerous search and training strategies

We are continuing to assess the impact of our collaborative research framework in enabling teacher-student collaboration. We derive a teacher architecture (T) and a student architecture (S) from the collaborative search procedure. As previously stated, the student architecture S is obtained by investigating searchable expansion ratios (U), yielding an FPS of 163 and a mIoU of 70%. On the other hand, direct compression of the teacher’s architecture with distillation training achieves only a mIoU of 66% and an FPS of 146. This demonstrates our architecture co-searching approach’s superiority over pruning-based compression strategies. We use knowledge distillation from a well-trained teacher architecture to improve the student’s accuracy further, resulting in a final FasterSeg network with

Methods	MIoU%	FPS	Resolutions
ENet	58.3	76.9	512 * 1024
ICNet	67.7	37.7	1024 * 2048
BiSeNet	69.0	105.8	768 *1536
FasterSeg	73.1	163.9	1024 * 2048

Table 4.2: Comparison of the models

an accuracy of 73.1%.

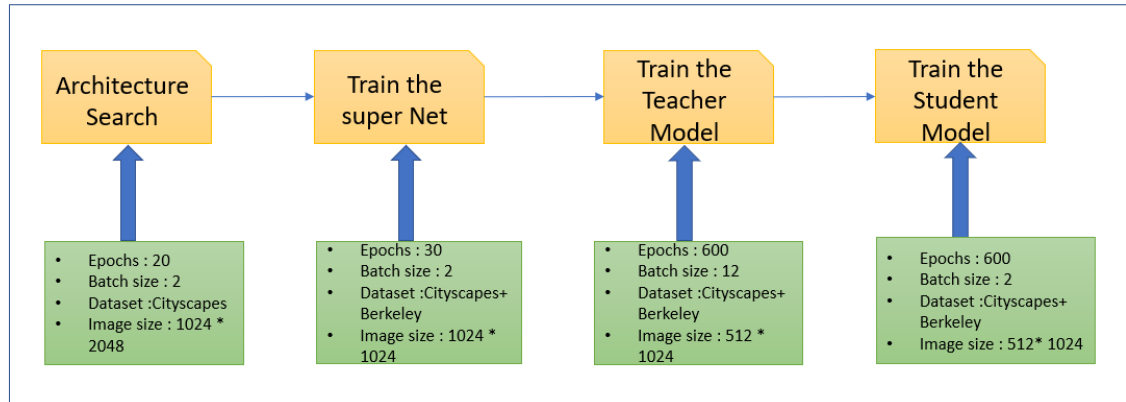


Figure 4.4: System overview with settings of parameters.

We conduct evaluations on the Cityscapes test and own dataset to assess the performance of FasterSeg. The mean Intersection over Union (mIoU) and inference speed are measured using the original image resolution 1024x2048. Our results, as presented in Table 4.1, demonstrate that FasterSeg achieves an impressive FPS of 163.9, even at the highest image resolution. This remarkable frame rate surpasses human-designed networks by more than 1.3 times. Also, it gives a better result than another model that only trained over the cityscapes dataset since the FasterSeg model learns all complex scenarios of the cityscapes and Berkeley dataset. The comparison is shown in table (4.2). Additionally, FasterSeg maintains competitive accuracy, achieving a mIoU of 73.1% on the validation set and 71.5% on the test set. When evaluated on our original dataset, the supernet architecture achieved an accuracy of 63.5%. It is worth noting that these high levels of precision are achieved solely through fine-annotated cityscapes images and another benchmarked Berkeley dataset without additional data.

4.6.3 Results

We utilized the FasterSeg model to conduct experiments on three distinct datasets, namely Berkeley, Cityscape, and our own dataset, encompassing various scenarios. Through this evaluation, we aimed to assess the model's performance and accuracy across different image datasets. In the first scenario, we examined an image (see figure 4.5) with low lighting conditions and a relatively simple surrounding environment, featuring objects such as traffic signs, pedestrians, and cars. We observed that the model successfully segmented (see figure 4.6) all the objects in the image and accurately predicted their respective classes. This indicates a promising performance of the model in this particular scenario.



Figure 4.5: Original Cityscape Image

In figure 4.7, an image taken from the Berkeley dataset under dim light conditions is presented. Upon examining the model's performance on this image, it is evident that several objects appear overlapped and incorrectly identified. The model (figure 4.8) struggles to effectively segment objects in areas with low light, resulting in compromised accuracy.

Furthermore, an image (figure 4.9) from our own dataset is showcased, capturing a scenario that includes small objects, cars, poles, people, buildings, and trees. The model's performance on the Indian driving scenario demonstrates notable improvement compared to the Berkeley dataset. In this case, the model successfully (figure 4.10) segments all the objects in the image, showcasing precise boundaries for each object. This indicates that the model performs well in accurately identifying and segmenting objects in the Indian driving scenario.



Figure 4.6: Result Of Cityscape image.



Figure 4.7: Original Berkeley image.

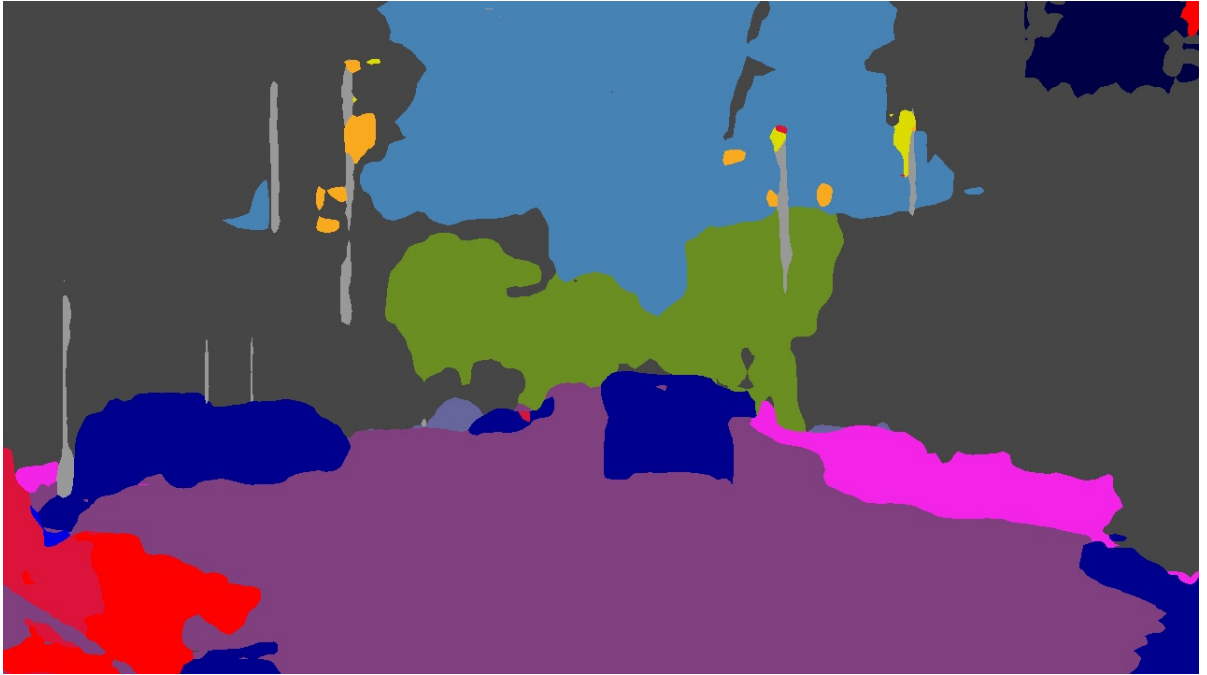


Figure 4.8: Result of Berkeley image.



Figure 4.9: Illustrates own dataset image.



Figure 4.10: Result of Own dataset image.

CHAPTER 5

Real-Time Experiments

We used the Raspberry Pi, initially released in 2012, is a lightweight single-board computer available in different models, including A, B, and Zero. This compact device encompasses a range of components that be explored in the subsequent sections. Beyond its size, the Raspberry Pi functions as a fully-fledged computer, capable of connecting peripherals such as a mouse, keyboard, and screen. It provides a user-friendly Linux desktop environment that can be effortlessly set up and configured. Moreover, the Raspberry Pi is not restricted to a single operating system and supports various lightweight OS options, offering versatility in its usage.

5.1 Edge Device

Further, edge computing, in general, focuses on processing and analysing data in close proximity to where it is generated or used. This technique has several advantages, including lower latency, higher data privacy, and increased durability in settings with limited or inconsistent network connectivity. By using the Raspberry Pi as an edge device, developers may harness the potential of local data processing, allowing for more efficient and responsive real-time apps, IoT deployments, and edge analytics. Components of Raspberry Pi are described below:

1. **General Purpose Input/Output (GPIO):** The GPIO is a key feature of the Raspberry Pi, allowing connections to various electronic components such as LEDs, motors, relays, and sensors. It facilitates both reading and transferring electronic signals between the Raspberry Pi and connected devices.
2. **Ethernet/USB/HDMI ports:** The Ethernet connector allows to connect to the network through a cable. A USB connector is also available for connecting a mouse, keyboard, web camera, and USB devices. Moreover, the HDMI connector allows the screen to be shown on a projector or monitor.

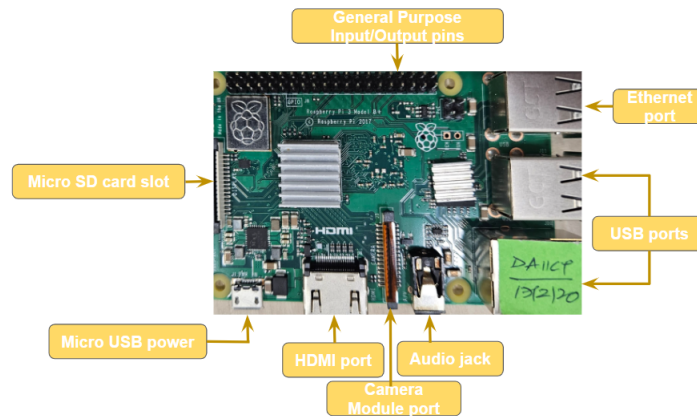


Figure 5.1: Raspberry Pi diagram labeled with various components. Resource: <https://www.raspberrypi.org>

3. **Audio Jack:** This component of the raspberry pie enables audio functioning. Connect headphones or a speaker to this component to make the system audible.
4. **Wi-Fi:** The Raspberry Pi also has Wi-Fi built in, enabling users to connect to wireless networks without a wired connection being required.
5. **Bluetooth:** The Raspberry Pi has built-in Bluetooth capability that enables users to connect to Bluetooth peripherals like speakers, headphones, and other audio equipment.
6. **Camera Module Port:** This port connects the Raspberry Pi camera. Do not connect the web camera since it can connect to a USB port.
7. **Micro USB power:** Raspberry pie necessitates a 5V steady power supply. The power supply is therefore attached to this port.
8. **Micro SD Card:** Micro SD card is used to store the data. Here, the SD Card serves as the bootable card by storing the operating system and enabling the raspberry pie board to boot from it. It also functions as a hard drive, storing all the users' private files.

5.2 Use Cases

The Raspberry Pi is a flexible and inexpensive computer that can be used for a range of applications. Here are some popular Raspberry Pi applications:

The Raspberry Pi is a flexible and inexpensive computer that can be used for a range of applications. Here are some popular Raspberry Pi applications:

- **Home Automation:** The Raspberry Pi is an ideal device for building a home automation system because of its inexpensive cost and great level of customization. It can be used with software such as Home Assistant to control lighting, measure the temperature, security systems, and other devices.
- **Robotics:** The Raspberry Pi is a generous system for building robots. It can also be programmed in languages like Python, C or used to operate motors, sensors, and other components.
- **Portable Computer:** With the addition of a battery, monitor, and keyboard, the Raspberry Pi can be smoothly transformed into a portable computer. This can be handy for people who desire a lightweight and inexpensive computer for basic operations like web surfing, document editing, and programming.
- **Education:** The Raspberry Pi was created to encourage computer science education, and it continues to be a popular tool for teaching programming and electronics. It can be used to teach basic programming ideas and more complex topics like robotics, machine learning, and artificial intelligence.
- **Network Traffic Analysis:** The Raspberry Pi is capable of running a virtual private network (VPN), performing security assessments, and monitoring network traffic. It also can block unwanted traffic using software like Nagios, Zabbix, or Pi-hole, which can monitor network activities.
- **Others:** The Raspberry Pi is widely used as a versatile and affordable camera and video recorder. It finds applications in various industries, research endeavors, and specialized fields. Our research incorporated additional circuits into the robot and integrated the Raspberry Pi as a camera module to capture diverse road images. The collected images are then sent to the controller node for segmentation and analysis. Moreover, this device has found utility in research, supporting areas such as farming and wildlife monitoring, plant phenotyping, underwater video surveillance, electronic sensing and control for studying animal behavior and physiology, bio-acoustics, autonomous learning experiments, and environmental monitoring.

The functionality of the Raspberry Pi's components is one of its main advantages. Numerous input/output interfaces are available, including HDMI, USB, Ethernet, Wi-Fi, Bluetooth, and GPIO pins. With the help of these interfaces, users can attach a wide range of gadgets and add-ons to the Raspberry Pi, including

keyboards, mouse, sensors, cameras, and motors. The Raspberry Pi is a flexible platform for several applications since it has an efficient processor and can run several operating systems, including Linux and Windows 10.

5.3 Demonstration

The Raspberry Pi has become an enormously popular tool for makers, enthusiasts, and professionals thanks to its combination of component functionality, powerful computing, and affordable cost. The Raspberry Pi offers the adaptability and power to turn vision into reality, whether developing a home automation system, a media center, or an autonomous car. The Raspberry Pi is likely to become an increasingly crucial element in the field of electronics and computers as it develops and gets better.

We describe the experimental setup and methodology used in our study to evaluate the performance of the proposed model in the context of autonomous driving. Our goal is to assess the effectiveness of the semantic segmentation model deployed on an edge device (Raspberry Pi) for real-time object detection and recognition.

The experimental environment consisted of several components, including a router for creating a local network, a web camera for capturing images of the surrounding environment, a system for monitoring and analyzing the results, and the edge device (Raspberry Pi) where the proposed model is deployed. All these components are interconnected using the Robot Operating System (ROS) framework, facilitating communication and data exchange.

To conduct the experiments, we treat the web camera as one node in the ROS graph, responsible for capturing images in real time. The deployed model is also considered a node, subscribing to the images generated by the web camera through a specific ROS topic. The proposed model then applied semantic segmentation on the received images to identify and classify objects within the scene.

Additionally, our system, which acts as another node in the ROS graph, subscribed to the results of the deployed model. This allows us to visualize and analyze the model's output, assessing the accuracy and performance of the semantic segmentation.

Throughout the experiments, we captured multiple videos of approximately five minutes each, encompassing different driving scenarios on the Gandhinagar city road. These videos served as input data for the semantic segmentation model, enabling us to evaluate its performance in real-world driving conditions.

By deploying the model on the edge device and integrating it with the ROS framework, we aim to build a robust and efficient autonomous system for object detection and recognition. The experiment section presents the details of our setup (figure:5.2), the data collection process, and the methodology followed to assess the performance of the proposed model in the autonomous driving context.

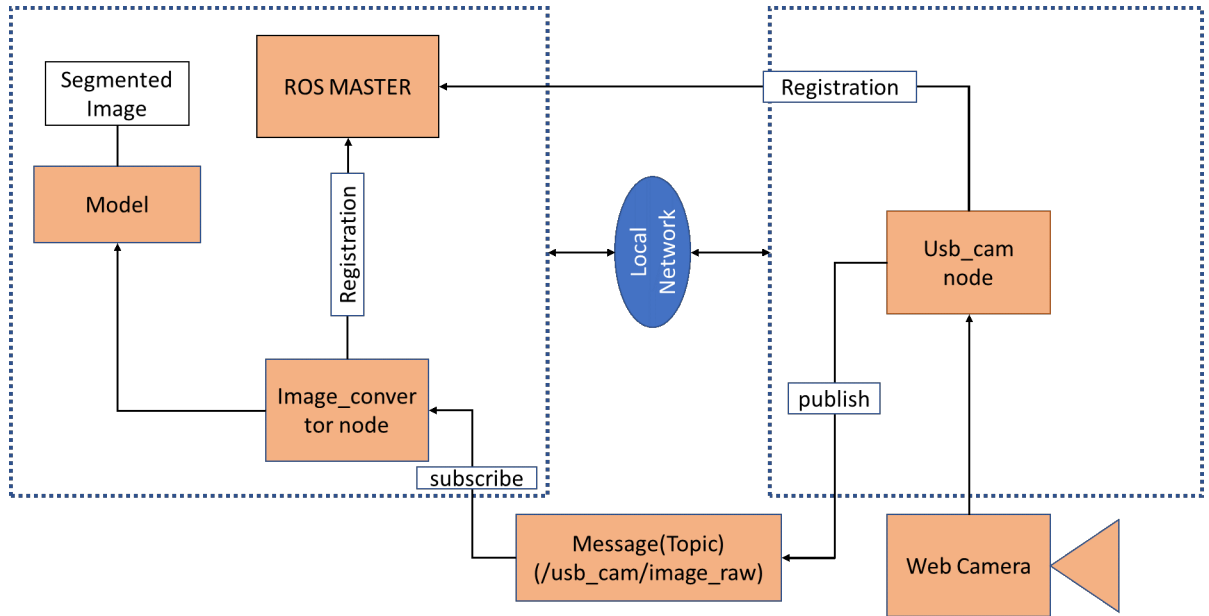


Figure 5.2: Illustrates the schematic diagram of ROS process for experiment setup.

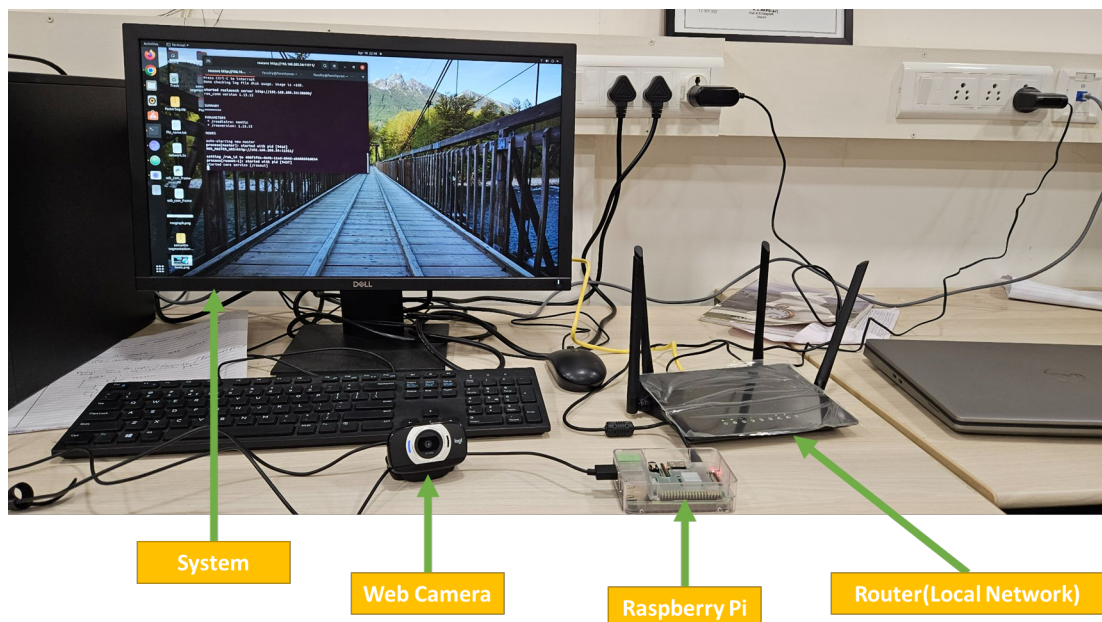


Figure 5.3: Demonstrates the real-time experiment performed.

CHAPTER 6

Conclusions

In this research, we concentrated on the object identification and recognition module within the context of the autonomous driving problem. We started with the YOLO architecture, famed for its real-time object detection capabilities. However, due to certain restrictions, we investigated alternate methods and discovered that semantic segmentation provided accurate object boundaries with low latency and high accuracy. We combined the Neural Architecture Search (NAS) technique with reinforcement learning to create a powerful deep network. This network was trained on a benchmarked cityscapes dataset and Berkeley dataset, allowing it to perform effectively in real-time segmentation tasks.

We used an edge device, a powerful system, the ROS framework, and a camera module to build a strong system. We have proved the usefulness of semantic segmentation in the context of object identification and recognition for autonomous driving in this study. The NAS-optimized deep network trained on the cityscapes dataset outperformed other deep networks in real-time segmentation. We designed a robust system capable of reliably identifying and recognizing objects in real-world driving scenarios by employing edge computing and interacting with the ROS framework.

This research sheds insight into using intelligent algorithms to solve real-world difficulties in autonomous driving.

References

- [1] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," *In Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3431–3440, 2015.
- [2] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng, et al., "ROS: an open-source robot operating system" *In ICRA workshop on open source software*, vol. 3, pp. 1-5, Kobe, Japan, 2009.
- [3] J. Fernandez, B. Allen, P. Thulasiraman, and B. Bingham, "Performance study of the robot operating system 2 with qos and cyber security settings," *In 2020 IEEE International Systems Conference (SysCon)*, pp. 1–6, 2020.
- [4] N. Koenig and A. Howard, "Design and use paradigms for Gazebo: an open-source multi-robot simulator," *In 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 3, pp. 2149–2154, 2004.
- [5] R. B. Rusu and S. Cousins, "3D is here: Point cloud library (PCL)," *In 2011 IEEE international conference on robotics and automation*, pp. 1–4, 2011.
- [6] I. Culjak, D. Abram, T. Pribanic, H. Dzapov, and M. Cifrek, "A brief introduction to openCV," *In 2012 proceedings of the 35th international convention MIPRO*, pp. 1725–1730, 2012.
- [7] H. R. Kam, S.-H. Lee, T. Park, and C.-H. Kim, "RVIZ: a toolkit for real domain data visualization," *Telecommunication Systems*, vol. 60, pp. 337–345, 2015.
- [8] R. Girshick, "Fast R-CNN," *In Proceedings of the IEEE international conference on computer vision*, pp. 1440–1448, 2015.
- [9] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," *In Proceedings of the IEEE international conference on computer vision*, pp. 2961–2969, 2017.
- [10] J. Dai, K. He, and J. Sun, "Instance-aware semantic segmentation via multi-task network cascades," *In Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3150–3158, 2016.

- [11] Y. Zhou, Y. Zhu, Q. Ye, Q. Qiu, and J. Jiao, "Weakly supervised instance segmentation using class peak response," *In Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3791–3800, 2018.
- [12] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "DeepLab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no.4, pp. 834–848, 2017.
- [13] A. Paszke, A. Chaurasia, S. Kim, and E. Culurciello, "ENet: A deep neural network architecture for real-time semantic segmentation," *arXiv preprint*, arXiv:1606.02147, 2016.
- [14] A. Chaurasia and E. Culurciello, "LINKNet: Exploiting encoder representations for efficient semantic segmentation," *In 2017 IEEE visual communications and image processing (VCIP)*, pp. 1–4, 2017.
- [15] H. Zhao, X. Qi, X. Shen, J. Shi, and J. Jia, "ICNET for real-time semantic segmentation on high-resolution images," *In Proceedings of the European conference on computer vision (ECCV)*, pp. 405–420, 2018.
- [16] C. Liu, L.-C. Chen, F. Schroff, H. Adam, W. Hua, A. L. Yuille, and L. Fei-Fei, "Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation," *In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 82–92, 2019.
- [17] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean, "Efficient neural architecture search via parameters sharing," *In International conference on machine learning*, pp. 4095–4104, 2018.
- [18] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy, "Progressive neural architecture search," *In Proceedings of the European conference on computer vision (ECCV)*, pp. 19–34, 2018.
- [19] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *In Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788, 2016.
- [20] J. Redmon and A. Farhadi, "YoLo9000: better, faster, stronger," *In Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7263–7271, 2017.

- [21] J. Redmon and A. Farhadi, "YoLov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.
- [22] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," *arXiv preprint*, arXiv:2004.10934, 2020.
- [23] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, and L. Antiga, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, pp. 32, 2019.
- [24] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," *In Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 580–587, 2014.
- [25] P. Jiang, D. Ergu, F. Liu, Y. Cai, and B. Ma, "A review of YoLo algorithm developments," *Procedia Computer Science*, vol. 199, pp. 1066–1073, 2022.
- [26] C.-Y. Wang, H.-Y. M. Liao, Y.-H. Wu, P.-Y. Chen, J.-W. Hsieh, and I.-H. Yeh, "CSPNet: A new backbone that can enhance learning capability of CNN," *In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, pp. 390–391, 2020.
- [27] S. Liu, L. Qi, H. Qin, J. Shi, and J. Jia, "Path aggregation network for instance segmentation," *In Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 8759–8768, 2018.
- [28] J. Hosang, R. Benenson, and B. Schiele, "Learning non-maximum suppression," *In Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4507–4515, 2017.
- [29] M. Cordts, M. Omran, S. Ramos, T. Scharwächter, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset," *In CVPR Workshop on the Future of Datasets in Vision*, vol. 2, 2015.
- [30] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," *In Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3213–3223, 2016.
- [31] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," *In Computer Vi-*

- sion–ECCV 2014: 13th European Conference, Zurich, Switzerland*, pp. 740–755, 2014.
- [32] W. Chen, X. Gong, X. Liu, Q. Zhang, Y. Li, and Z. Wang, "Fasterseg: Searching for faster real-time semantic segmentation," *arXiv preprint*, arXiv:1912.10917, 2019.
- [33] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *The Journal of Machine Learning Research*, vol. 20, no. 1, pp. 1997–2017, 2019.
- [34] J. Gou, B. Yu, S. J. Maybank, and D. Tao, "Knowledge distillation: A survey," *International Journal of Computer Vision*, vol. 129, pp. 1789–1819, 2021.
- [35] A.-C. Cheng, C. H. Lin, D.-C. Juan, W. Wei, and M. Sun, "Instanas: Instance-aware neural architecture search," *In Proceedings of the AAAI conference on artificial intelligence*, vol. 34, pp. 3577–3584, 2020.
- [36] C. Yu, J. Wang, C. Peng, C. Gao, G. Yu, and N. Sang, "Bisenet: Bilateral segmentation network for real-time semantic segmentation," *In Proceedings of the European conference on computer vision (ECCV)*, pp. 325–341, 2018.
- [37] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," *In Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4510–4520, 2018.
- [38] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, "Encoder-decoder with atrous separable convolution for semantic image segmentation," *In Proceedings of the European conference on computer vision (ECCV)*, pp. 801–818, 2018.
- [39] J. Yu, L. Yang, N. Xu, J. Yang, and T. Huang, "Slimmable neural networks," *arXiv preprint*, arXiv:1812.08928, 2018.
- [40] D. Stamoulis, R. Ding, D. Wang, D. Lymberopoulos, B. Priyantha, J. Liu, and D. Marculescu, "Single-path NAS: Designing hardware-efficient convnets in less than 4 hours," *In Machine Learning and Knowledge Discovery in Databases: European Conference, Würzburg, Germany*, pp. 481–497, 2020.
- [41] Y. Zhang, Z. Qiu, J. Liu, T. Yao, D. Liu, and T. Mei, "Customizable architecture search for semantic segmentation," *In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11641–11650, 2019.

- [42] A. Shaw, D. Hunter, F. Landola, and S. Sidhu, "Squeezenas: Fast neural architecture search for faster semantic segmentation," *In Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, pp. 1–11, 2019.