DE-42 (MTS) ARHAM, ZAIN

SEMANTIC SCENE CLASSIFICATION USING AI WITH CAMERA FOR AUTONOMOUS VEHICLES

COLLEGE OF
ELECTRICAL AND MECHANICAL ENGINEERING
NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY
RAWALPINDI
2024

COLLEGE OF ELECTRICAL AND MECHANICAL ENGINEERING

DE-42 MTS
PROJECT REPORT

SEMANTIC SCENE CLASSIFICATION USING AI WITH CAMERA FOR AUTONOMOUS VEHICLES

Submitted to the Department of Mechatronics Engineering
in partial fulfillment of the requirements
for the degree of
Bachelor of Engineering
in
Mechatronics
2024

Sponsoring DS:

Dr. Tahir Habib Nawaz (Supervisor)
Dr. Ayesha Zeb (Co-Supervisor)

Submitted By:

Muhammad Arham Imran
Muhammad Zain ul Abedeen

ACKNOWLEDGMENTS

To conduct this thesis on the perception module of self-driving cars, we would like to express our gratitude to Allah Almighty, who bestowed His blessings to carry out this extensive research. Further, we would like to extend our humble gratitude to our supervisor, Dr. Tahir Habib Nawaz, whose valuable advice helped us surmount many hurdles. We are also thankful to our co-supervisor, Dr. Ayesha Zeb, for providing great assistance. Finally, a huge thanks to our parents, friends, and colleagues for their support.

ABSTRACT

The fundamentals of an autonomous vehicle are perception of the environment, object detection, path planning, and control systems for vehicle actuation. There are two levels of perception, high-level scene classification and low-level depth perception. This thesis targets the high-level scene classification for an autonomous vehicle.

High-level scene classification means to classify each pixel in a captured scene with an object class. With the advancement in technology and the rise of Artificial Intelligence, semantic segmentation models were developed performing this task. Hence among these models, Segformer was employed in this project for scene classification on a pixel level. A live video feed will be supplied to the model through a camera for classification. An IoT control has also been implemented to demonstrate the objective of providing a failsafe control switchover to a human in case of any complication.

To classify the environment in real time so to be suitable for an autonomous vehicle, the entire AI algorithm was ported onto an embedded platform, the Jetson Nano, which is designed to manage the computational requirements of deep learning algorithms. The model was able to run at a mean of 1.71 fps on the Jetson Nano. The failsafe control was implemented using IoT over Wi-Fi on an ESP32.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

Acronyms

LCD Liquid Crystalline Display
HDMI High-Definition Multimedia Interface
CSI Camera Serial Interface
IoT Internet of Things
SBC Single Board Computer
INT8 8-bit Integer
FP16 16-bit Floating Point
Iou Interaction over union

Chapter 1 - INTRODUCTION

1.1. Overview:

In recent years, the advancement of technology, particularly in artificial intelligence (AI), has catalyzed a profound transformation in the automotive industry with the emergence of autonomous vehicles. These vehicles, commonly referred to as self-driving cars, represent a paradigm shift in transportation, offering the potential for safer, more efficient, and accessible mobility solutions. The integration of sophisticated sensor systems, including cameras, radar, lidar, GPS, and advanced computing technology, enables autonomous vehicles to perceive their environment and navigate autonomously.

1.2. Motivation:

The motivation driving this research stems from recognizing the transformative potential of autonomous vehicles in reshaping transportation systems worldwide. The prospect of reducing traffic accidents, congestion, and emissions while enhancing mobility for individuals with disabilities and the elderly underscores the urgency of advancing autonomous driving technology. Central to realizing these benefits is the development of robust perception systems that enable vehicles to accurately interpret their surroundings in diverse and dynamic environments. Perception forms the cornerstone of autonomous driving, allowing vehicles to recognize and respond to obstacles, road signs, pedestrians, and other vehicles in real-time.

1.3. Problem Formulation:

At the core of our research lies the intricate challenge of developing perception systems capable of operating reliably in complex and unpredictable environments. This entails overcoming various technical hurdles, including the accurate detection and classification of objects, robust tracking of their movements, and precise localization within the vehicle's surroundings. Furthermore, ensuring the scalability

and adaptability of perception algorithms across different driving conditions poses additional challenges. To address these issues, our research focuses on leveraging cutting-edge computer vision techniques, particularly semantic scene understanding, to enhance the perceptual capabilities of autonomous vehicles.

1.4. Background Knowledge:

With current advancement in the in the technology and research especially in the field of AI, Autonomous vehicle industry has seen a significant rise as a result. Autonomous vehicles, commonly known as self-driving cars, rely on a sophisticated network of sensors, cameras, radar, lidar, GPS, and advanced computer systems to operate[1]. These sensors include cameras for capturing images and videos, radar for detecting objects and determining distances, and lidar for creating precise 3D maps using laser technology. The onboard computers/embedded system process this sensor data in real-time, using advanced machine learning and artificial intelligence algorithms to identify objects, pedestrians, other vehicles, road signs, and road conditions. Based on this data analysis, autonomous vehicles make instantaneous decisions regarding navigation, including speed, acceleration, braking, lane changes, and route planning. Safety is paramount in autonomous vehicle development, with redundant systems in place to reduce the risk of accidents. The level of autonomy in these vehicles varies, with most current models falling within the range of Level 2 to Level 4, indicating their capacity to perform some or most driving tasks under specific conditions[2]. However, challenges persist, including navigating complex environments, ethical dilemmas in decision-making during critical situations, and ensuring robust cybersecurity. Despite these challenges, the development of autonomous vehicles has the potential to revolutionize transportation, enhancing safety, efficiency, and accessibility. Our main interest lies in the development of perception for autonomous vehicles. Perception in autonomous vehicles is the essential sensory function that allows them to understand and engage with their environment. By processing data from diverse sensors like cameras, radar, lidar, and ultrasonic

sensors, these vehicles can identify and categorize objects, assess their positions and predict their movements. Object detection and lane recognition play a critical role, enabling the identification of vehicles, pedestrians, and lane markings. Continuous advancements in sensor technology and algorithms are driving improvements in perception, making autonomous driving safer and more dependable. In this paper, we will be overviewing high level perception for autonomous vehicles through computer vision by employing semantic scene classification for a better understanding of the scene by the autonomous vehicles. Semantic scene classification, a critical computer vision task, is at the heart of developing autonomous vehicles. It involves labeling each pixel in an image with a specific object category or class[3]. This technology allows autonomous vehicles to perceive their surroundings, identify road elements, obstacles, and other vehicles, and make informed decisions based on this understanding. In this comprehensive literature review, we delve into the history, recent breakthroughs, challenges, and future directions of semantic scene classification for autonomous vehicles.

1.5. Contribution:

This project aims to set up a place for studying self-driving cars. We want to create a platform where we can try out different ideas and add new features to see how they work. It is a further modification of the previous project for self-driving cars which was environmental depth perception for autonomous vehicles. It is our effort to try this project in the Mechatronics Engineering Department at the College of Electrical and Mechanical Engineering, NUST so it lays basis for the further advancement of autonomous vehicles in the department.

1.6. Organization of the Thesis:

This thesis is further written in the following fashion:

Chapter 2: Provides a Literature Review regarding the existing problems in the scenery classification for autonomous vehicles and existing methodologies for

developing effective and robust AI models.

Chapter 3: Presents the details of methodology employed by our team in completing this project and explains the inner working of the project.

Chapter 4: Briefly discusses the summary of results, findings, and comparisons of the model.

Chapter 5: Concludes the report and explores future possibilities and directions in which the project can be taken.

Chapter 2: LITERATURE REVIEW

Semantic scene classification for autonomous vehicles has undergone significant historical development over the years, reflecting advances in computer vision, machine learning, and sensor technologies. This historical overview provides a detailed account of the key milestones and trends in this field.

2.1. Early Approaches (2000s):

Semantic scene classification for autonomous vehicles began with early approaches in the 2000s. These methods relied on rule-based algorithms and basic image processing techniques. They focused on simple tasks like lane detection and obstacle recognition. At this stage, the emphasis was on developing reliable algorithms for vehicle guidance and safety. An example of such algorithms is the linear Hough Transform (HT) which is a popular algorithm for line detection and is also widely employed for the purpose of lane detection[4]. HT operates through "voting" and "peak detection" steps. During voting, edge pixels are transformed into sinusoidal curves based on their coordinates (x, y). The resulting $\rho-\theta$ values, representing lines, are accumulated in a 2-D array. Peaks in this array indicate straight lines in the image. Peak detection analyzes the array to identify these lines. More about HT can be found here[5].

Another technique was the Sliding Window Technique, extensively utilized in computer vision for lane detection, operates through a systematic process. It begins at the image's base, computing a histogram of pixel intensities along the horizontal axis to identify potential starting points for lane lines. Subsequently, sliding windows are placed around these peaks, moving vertically upward to trace the lane lines. Within each window, the algorithm uses techniques like the Canny edge detector to refine lane positions. Canny edge detection identifies object boundaries by detecting rapid inten2sity changes in an image; It computes gradients, applies non-maximum suppression, and tracks edges, resulting in a binary image

highlighting detected edges accurately and minimizing noise[6]. Sliding window's iterative method adjusts window placement to match the lane lines' shape and curvature, enhancing accuracy. The sliding window technique was designed to structure time-series data. Implementations created at the time, such as the Static Sliding Window (SSW), maintain a fixed window length over time, ensuring that time-series data points occur at consistent intervals[7].

Other than lane detection techniques, there were also object detection techniques introduced in this era, among them being Histogram of Oriented Gradients (HOG)[8] and Cascade Classifiers[9]. HOG is a feature descriptor used in computer vision and image processing. It captures local object shape information in an image. HOG works by dividing the image into small cells, computing gradients within each cell, and then creating histograms of gradient orientations. These histograms represent the distribution of edge orientations in the image. HOG is particularly useful for object detection tasks, such as real-time pedestrian detection[8] and vehicle detection[10], as it can capture the shape and appearance of objects in varying lighting conditions and scales.

A cascade classifier is a machine learning object detection framework tailored for efficiently identifying vehicles within images or video streams. It functions by applying a series of specialized classifiers in a sequential manner, with each stage working to swiftly dismiss non-vehicle regions while retaining true positive detections. This cascading strategy enables the classifier to rapidly eliminate most image areas that do not contain vehicles, thereby reducing computational demands and expediting the detection process. Cascade classifiers are especially well-suited for real-time applications like vehicle detection[9] [11], where quick and accurate identification of vehicles is critical. Their efficacy lies in their capacity to make prompt decisions, allowing the model to avoid unnecessary computations on areas of the image that are improbable to contain vehicles.

2.2. Deep Learning Revolution (2010s):

The field of semantic scene classification underwent a transformative shift with the advent of deep learning, especially Convolutional Neural Networks (CNNs). Researchers recognized the potential of deep learning models to significantly improve segmentation accuracy. The application of CNNs for pixel-level classification brought about a paradigm shift. In 2015, Long et al. introduced the Fully Convolutional Network (FCN)[12], a pioneering architecture designed for end-to-end semantic segmentation. This was a critical breakthrough that laid the foundation for modern semantic scene classification.

Fully Convolutional Networks (FCN) replaced fully connected layers with convolutional layers to maintain spatial information. FCN employs an encoder-decoder architecture, where the encoder extract features from the input image, and the decoder upscales the features to generate a pixel-wise prediction map. Skip connections are used to combine feature maps from different encoder layers, allowing the network to capture multi-scale information and enhance segmentation accuracy[12].

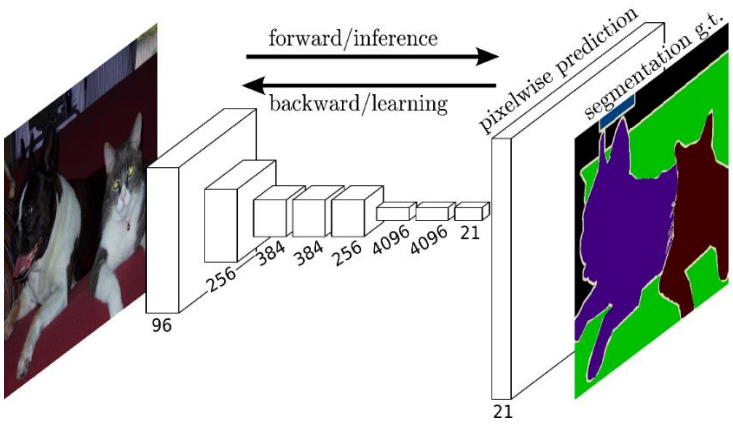

Figure 1: FCN Architecture

The Conditional Random Field Recurrent Neural Network (CRF-RNN)[13], introduced in 2015, is a model that combines a Convolutional Neural Network (CNN) for pixel-wise predictions with a dense Conditional Random Field (CRF)

for post-processing. It uses a Recurrent Neural Network (RNN) to iteratively refine and improve the results of semantic segmentation by considering spatial relationships between pixels. CRF-RNN is trained end-to-end and is applied as a post-processing step to enhance segmentation accuracy, making it particularly valuable in scenarios where fine details and contextual information are important.

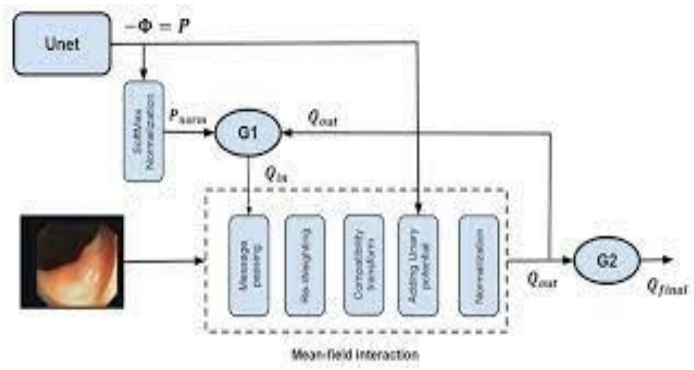

Figure 2. CRF-RNN Architecture

ParseNet[14], introduced in 2016, is a semantic segmentation model that emphasizes multi-scale context aggregation. It leverages spatial pyramid pooling to gather information from different receptive fields, enhancing segmentation accuracy. ParseNet provides comprehensive scene understanding and is known for its ability to capture fine details in images.

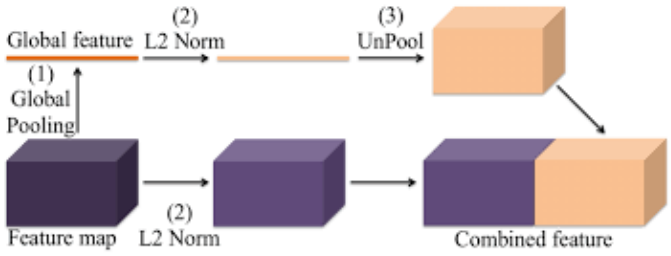

Figure 3: ParseNet Architecture

PSPNet[15], introduced in 2016, is a semantic segmentation model that focuses on multi-scale context aggregation using spatial pyramid pooling. It improves segmentation accuracy by incorporating information from various receptive fields, making it effective in capturing scene detail.

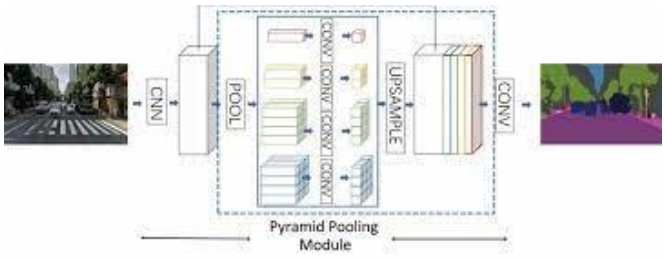

Figure 4: PSPNet Architecture

2.3. Further Model Architecture and Refinements (2010s):

The 2010s and early 2020s witnessed the development of various model architectures specifically tailored for semantic scene classification. Models like UNet, DeepLab, and ReSeg became popular choices.

U-Net is particularly popular in medical imaging and applications where precise segmentation is crucial. It features both a contracting path (encoder) and an expansive path (decoder), allowing it to capture fine-grained details. The use of skip connections connecting encoder and decoder layers helps maintain spatial information and improve segmentation accuracy[16].

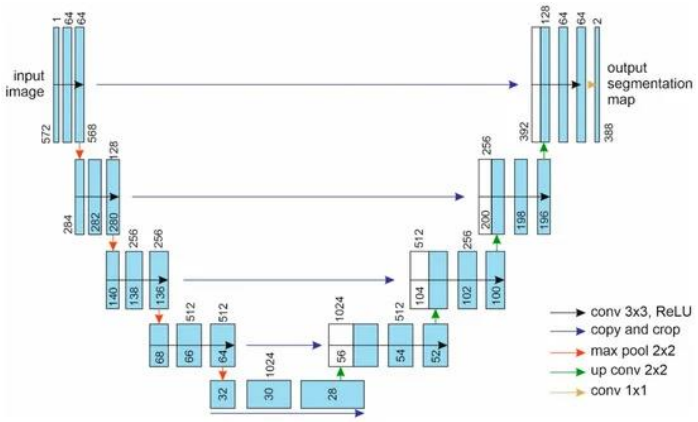

Figure 5: UNet Architecture

DeepLab models leverage atrous (dilated) convolutions to capture multi-scale information effectively. They also employ a spatial pyramid pooling module to gather information from various receptive fields, leading to high segmentation accuracy. DeepLab models find applications in scenarios like autonomous driving

and remote sensing where accuracy is paramount[17].

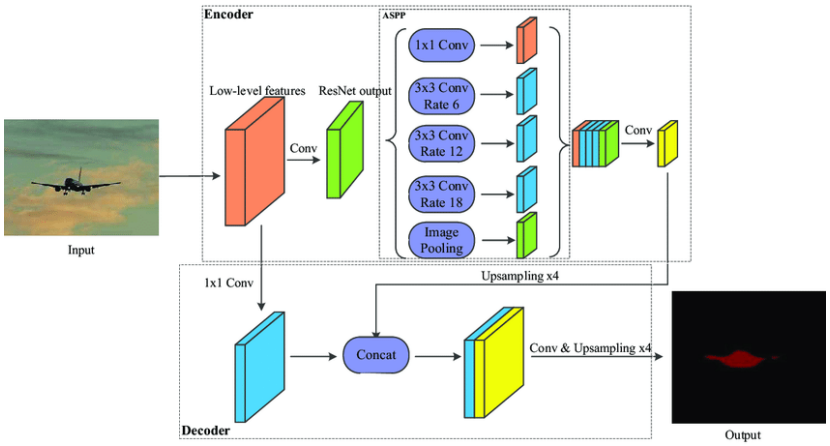

Figure 6. DeepLab Architecture

ReSeg[18], introduced in 2016, is a semantic segmentation model that employs recurrent neural networks (RNNs) to iteratively refine and enhance pixel-level object labeling in images and video frames. This unique approach enables the model to capture intricate pixel dependencies and complex relationships, making it valuable in scenarios where objects are partially occluded or fine details must be preserved. ReSeg's iterative refinement process results in highly accurate and visually appealing segmentation maps, and it finds applications in fields such as medical image analysis, autonomous driving, and remote sensing, where precision in object delineation and scene understanding is essential for informed decision-making.

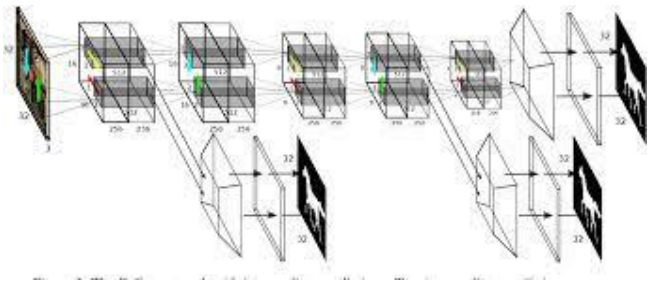

Figure 7. ReSeg Architecture

2.4. Sensor Fusion and Multimodal Perception (2010s – 2020s):

Autonomous vehicles rely on multiple sensors, including cameras, lidar, and radar.

The fusion of data from these sensors has become a significant trend in semantic scene classification research[19]. Researchers sought to integrate information from various sources to enhance segmentation accuracy and robustness in diverse environments. Sensor-fusion networks emerged as a solution to seamlessly combine data from different sensors.

2.5. Real-Time Processing and Efficiency:

The real-time processing requirement for autonomous vehicles necessitated the development of lightweight segmentation models that could maintain high accuracy while improving computational efficiency [20]. Researchers focused on optimizing model architectures, quantization techniques, and hardware acceleration to ensure that semantic scene classification could be performed with low latency and minimal power consumption.

Efficient Neural Network (ENet) is another model tailored for real-time applications, prioritizing efficiency without compromising segmentation accuracy. ENet uses a compact architecture and employs techniques such as spatial dropout and batch normalization to achieve real-time performance while maintaining adequate accuracy[21].

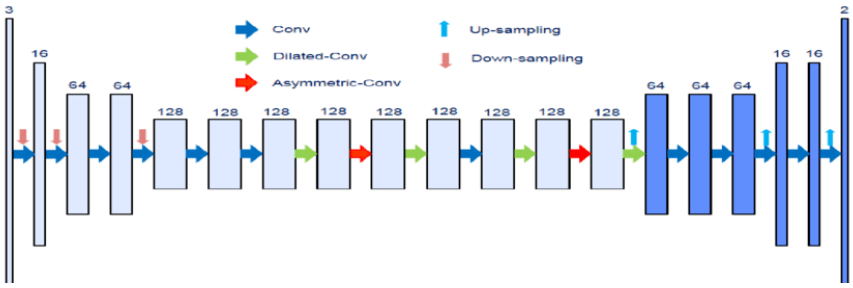

Figure 8. E-Net Architecture

Segmenter[22] is a semantic segmentation model introduced in 2019 that excels in providing precise pixel-level object labeling in images and video frames. What sets it apart is its efficiency and real-time processing capabilities, making it a valuable tool in applications like autonomous driving, robotics, and medical image analysis. This model leverages dilated convolutions, also known as atrous convolutions, to

capture multi-scale contextual information, ensuring that it can accurately segment objects of various sizes and contexts. Additionally, it incorporates spatial pyramid pooling, allowing it to gather information from different scales and receptive fields. The result is a model optimized for low-latency, on-the-fly segmentation, ideal for autonomous vehicles that need rapid object recognition and response. Segmenter's efficiency and real-time processing abilities make it a versatile solution for pixel-level object labeling in scenarios where quick and accurate semantic segmentation is essential for informed decision-making and automation.

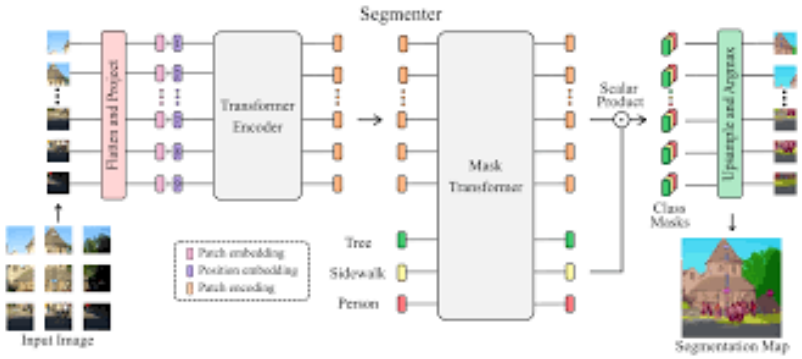

Figure 9:Segmenter Architecture

BiSeNet[23], short for Bilateral Segmentation Network, is a real-time semantic segmentation model introduced in 2018. This model stands out for its efficient and rapid pixel-level object labeling in both images and videos. BiSeNet utilizes a dual-branch architecture, where one branch captures global context from low-resolution feature maps, and the other focuses on local details from high-resolution feature maps. This design allows the model to achieve a balance between segmentation accuracy and real-time performance, making it well-suited for applications like autonomous driving, robotics, and augmented reality. Its ability to understand visual scenes quickly and accurately has made it invaluable in scenarios where timely decision-making and scene understanding are of paramount importance.

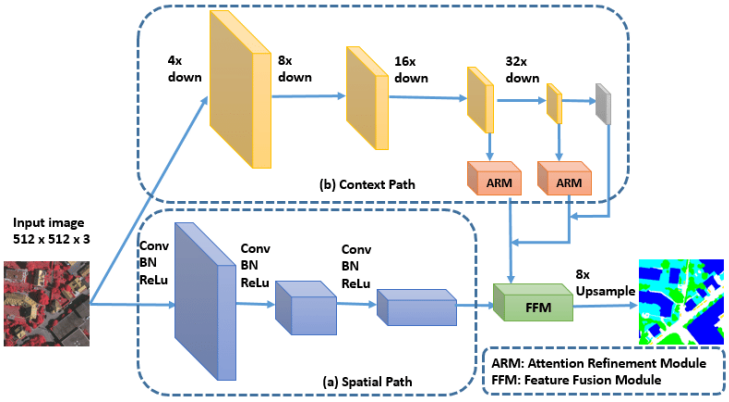

Figure 10: BisNet Architecture

2.6. Model Optimization:

Model optimization for embedded platforms is a critical consideration when deploying machine learning models on resource-constrained devices like smartphones, edge devices, IoT devices, and embedded systems[24]. The goal of optimization is to ensure that these models can execute efficiently, with a minimal memory footprint and improved energy efficiency. Several key techniques and aspects come into play in the process of model optimization for embedded platforms.

Quantization is a fundamental technique that reduces the bit-width of model weights and activations. This typically involves transitioning from 32-bit floating-point values to 8-bit integers. By reducing precision, quantization significantly lowers memory and computation requirements, rendering the model more compatible with embedded platforms. There are two primary approaches: post-training quantization, which quantizes a pre-trained model, and quantization-aware training, which trains models to be quantization-friendly from the outset[25].

Pruning is another valuable optimization technique that involves removing unimportant or low-magnitude weights from the model. The objective is to reduce the model's size and computation requirements without significantly compromising accuracy. Techniques like magnitude-based pruning and structured pruning are

widely used to achieve these objectives.

2.7. Embedded Systems Platform:

Embedded system platform for AI deployment encompasses a variety of technologies that are integral to enabling artificial intelligence (AI) capabilities at the edge, where devices can process data locally, without constant reliance on cloud servers. Here's a more detailed exploration of these technologies:

2.7.1. Single-Board Computers (SBCs):

Raspberry Pi SBCs[26] are affordable and versatile, making them popular for AI prototyping and development. Models like the Raspberry Pi 4 have increasingly powerful processors and GPU capabilities, making them capable of running AI workloads efficiently.

2.7.2. Field-Programmable Gate Arrays (FPGAs):

Xilinx and Intel (formerly Altera): Xilinx[27] and Intel[28] provide FPGAs that are widely used for AI acceleration. FPGAs are known for their flexibility, allowing users to program them to perform specific AI tasks efficiently. They excel in scenarios where customization and real-time processing are critical.

Intel Arria 10 FPGA: The Intel Arria 10 FPGA[28] is particularly recognized for its role in AI inference acceleration. It offers high performance and low power consumption, making it suitable for edge devices requiring real-time AI processing, such as autonomous vehicles and robotics.

2.7.3. AI Accelerator Chips:

Google Edge TPU: Google's Edge TPU[29] is designed for AI acceleration

on edge devices. It provides high-performance AI processing while ensuring low power consumption. The Edge TPU has been integrated into products like the Coral Dev Board and is suitable for tasks like object detection and image recognition.

NVIDIA Jetson Series: NVIDIA's Jetson[30] platform includes a range of devices, such as the Jetson Nano, Jetson Xavier, and Jetson AGX Xavier, all optimized for AI and robotics applications. These platforms integrate GPUs and NPUs to enable AI processing at the edge, making them ideal for robotics, drones, and other AI-driven projects.

2.7.4. Microcontrollers and Microprocessors:

ARM Cortex-M and Cortex-A Series: ARM-based microcontrollers and microprocessors are fundamental to edge AI in IoT devices. They offer a balance of performance and energy efficiency, making them suitable for AI applications like sensor data analysis, voice recognition, and predictive maintenance in smart devices.

NXP i.MX Series: NXP's i.MX processors are tailored for embedded vision and AI applications. These processors are used in various industries, including automotive, where they enable object recognition, machine learning, and human-machine interface technologies.

2.7.5. ASICs (Application-Specific Integrated Circuits):

Apple's Neural Engine: Apple's custom-designed AI accelerator ASIC[31], known as the Neural Engine, is integrated into iPhones and iPads. It optimizes performance and power efficiency for AI tasks, allowing for applications like image recognition and natural language processing.

2.7.6. Edge AI Development Kits:

NXP Vision Toolbox: NXP's Vision Toolbox is a comprehensive solution for AI acceleration on its i.MX processors. This toolbox allows edge devices to perform AI tasks such as object tracking and facial recognition, making it valuable for applications in robotics and industrial automation.

Intel Neural Compute Stick: This AI development kit features an Intel Movidius Myriad X VPU (Vision Processing Unit). It is designed for edge AI inference and enables a wide range of vision-based applications, including security cameras, drones, and smart retail systems.

2.7.7. Custom Hardware Solutions:

Some companies develop custom hardware solutions to meet the specific demands of their edge AI applications. These custom solutions may include AI accelerators, specialized SoCs, and hardware that align with their unique requirements, often in industrial or medical settings.

Embedded hardware for AI deployment continues to evolve and diversify, offering a wide array of options for different use cases and requirements. These hardware solutions enable edge devices to perform AI inference efficiently, contributing to advancements in various fields, including autonomous robotics, IoT, smart cities, and industrial automation.

2.8. Mobile Platforms:

The fundamental mechanical components of autonomous vehicles are integral to the evolution of self-driving technology. Key elements in a platform for autonomous vehicles include sensors, actuators, and control systems. Sensors, utilizing technologies like LiDAR, cameras, and radar systems, capture and interpret the vehicle's surroundings; Actuators, such as motors and servos, translate digital decisions into physical movements, facilitating functions like steering,

acceleration, and braking and control systems, often employing advanced algorithms and artificial intelligence, coordinate the interaction between sensors and actuators, enabling real-time decision-making. Additionally, this exploration addresses the challenges related to mechanical integration, underscoring the necessity for durability, resilience, and fault tolerance to ensure the vehicle's reliability in various environments. We were able to review a few prototypes, one of which was developed for a MIT hackathon in 2016 (initially developed in 2015)[32] and one is a prototype from Quanser (a company that is known for research on self-driving cars)[33], as the pictures show below:

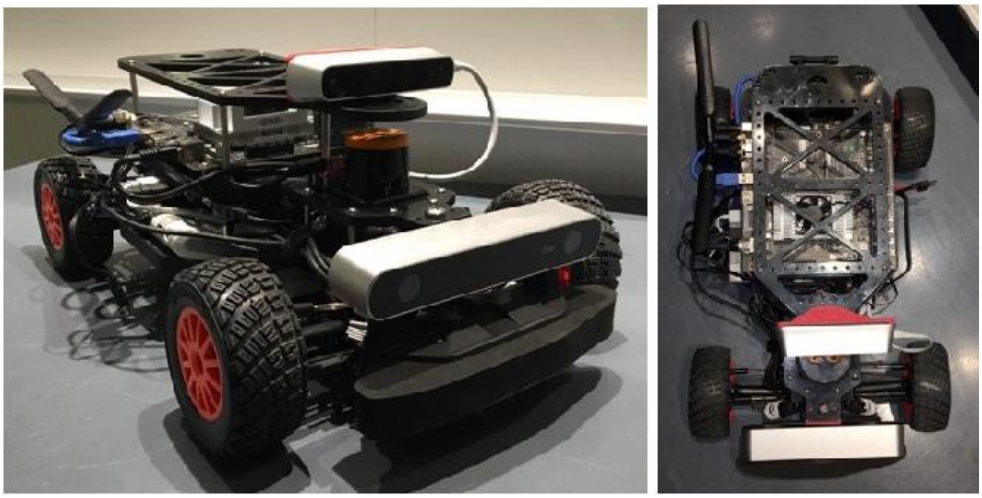

Figure 11. MIT Racecar Platform for January 2016 Hackathon

Figure 12. Qcar Developed by Quanser

2.9 Benchmark Datasets:

To benchmark and compare various segmentation algorithms, the creation of large, labeled datasets became essential. Notable datasets like Cityscapes, KITTI, and Mapillary Vistas provided a standardized platform for evaluating and advancing semantic scene classification techniques[34]. Research competitions, such as the Cityscapes and Semantic Segmentation Challenges, encouraged collaboration and the development of cutting-edge algorithms. All these datasets contain images that cover diverse situations and weather conditions that are very essential for the robustness of the model. The details of these datasets are summarized in the table as follows:

Table 1: Dataset Specifications

Dataset	Resolution	Classes	Number of Images	Release Data
Cityscape[35]	1024x2048 pixels.	20 different classes	5000 fine and around 20000 coarse annotated images	2016
Mapillary vista[36]	4000+x3000+ pixels	60 different classes	approximately 25,000 high resolution images	2017
DUS[37]	1024x440 pixels	DUS is a diverse dataset that labels a wide	DUS includes tens of thousands of images,	2021
Camvid[38]	720x960 pixels	11 different classes	701 images	2007
KITTI[39]	1242x375 pixels	It includes labels for a limited set of classes, primarily focusing on road and vehicle related objects.	around 7,500 labeled images for various tasks, including semantic scene classification.	2012

2.10 Summary:

The historical development of semantic scene classification for autonomous vehicles showcases a transition from rule-based algorithms to deep learning-based approaches. The integration of sensor data, benchmark datasets, and model refinements has led to significant improvements in accuracy and real-time processing, making this technology a fundamental component of safe and reliable autonomous driving systems. In this research we will look to aid in the development of an autonomous vehicle by implementing one of the latest neural network models to classify objects at the scene of the vehicle. The following chapter provides the different tools and description of their methods used in this project.

Chapter 3 – METHODOLOGY

3.1. Overview:

This chapter delineates the methodology that was adopted to perform this project, touching down on every important aspect of the project. Firstly, it will introduce the sensors, embedded platform and mobile base that were used in this project. Then the specification of them is explained further. Following that the software used to implement the model and the IoT network is explained. Finally, it will explain the detailed explanation and procedure for the implementation of semantic scene classification model from the data acquired from the Cityscape dataset and its deployment to the embedded platform along with the IoT network control to use it on an autonomous vehicle.

3.2. Hardware Specifications

3.2.1. Raspberry PiCam V2.1:

Figure 13: Raspberry PiCam V2.1

We are using a camera in this project as a sensor. The focal length of a camera lens determines the perspective and magnification of the captured image. Light enters the camera through the lens and is refracted to focus onto the image sensor. A shorter focal length means the lens brings the light

rays to focus closer to the image sensor, resulting in a wider field of view and less magnification. Conversely, a longer focal length brings the light rays to focus farther away from the image sensor, resulting in a narrower field of view and greater magnification. This characteristic allows photographers to adjust their composition, zooming in on distant subjects with longer focal lengths or capturing expansive scenes with shorter focal lengths.

The Raspberry Pi Camera Module V2.1 is a versatile and compact camera designed specifically for use with Raspberry Pi boards. It features a Sony IMX219 8-megapixel sensor, capable of capturing high-resolution images and videos. With its small form factor and lightweight design, the camera module can be easily integrated into various projects, from DIY surveillance systems to robotics and drones. The v2.1 camera module supports a wide range of resolutions and frame rates, allowing users to customize their capture settings based on their specific requirements. Additionally, it offers features such as autofocus and programmable control over parameters like exposure, white balance, and ISO sensitivity, providing flexibility and control to users. It uses a CSI connector to interface with other boards. Especially in terms of deep learning and computer vision projects, the Raspberry Pi Camera Module v2.1 offers a cost-effective and user-friendly solution for adding imaging capabilities to projects.

The specifications of camera are stated below:

Table 2. PiCam Specifications

Specification	Details
Sensor	Sony IMX219 8-megapixel sensor
Resolution	3280 × 2464 pixels

Frame Rate	1080p at 30fps, 720p at 60fps, and VGA at 90fps
Sensor Size	1/4 inch
Pixel Size	1.12 µm
Aperture	f/2.0
Lens	Fixed focus
Field of View	62.2 degrees (h), 48.8 degrees (v), 68.2 degrees (d)
Video Formats	RAW, RGB, YUV422, JPEG
Dimensions	25mm x 23mm x 9mm (approx.)
Weight	3 grams (approx.)
Power Requirement	250mA at 3.3V

3.2.2. Jetson Nano:

The NVIDIA Jetson Nano is a compact, yet powerful computer tailored for embedded applications and AI development. At its core is a NVIDIA Maxwell GPU boasting 128 CUDA cores, engineered for parallel processing, thus making it ideal for executing deep neural networks and other AI algorithms with remarkable efficiency. This GPU is complemented by a quad-core ARM Cortex-A57 CPU running at 1.43 GHz, furnishing ample processing power for diverse computing tasks. With 4GB of LPDDR4 RAM onboard, the Jetson Nano ensures smooth operation while running AI models and applications. Its compatibility with renowned frameworks such as TensorFlow, PyTorch, and OpenCV empowers

developers to seamlessly deploy and optimize their AI solutions.

Figure 14. Jetson Nano B01

In terms of connectivity, the Jetson Nano offers an array of options including Gigabit Ethernet, USB 3.0, USB 2.0, HDMI, and a MIPI-CSI camera interface, facilitating effortless integration with peripherals, cameras, displays, and networking devices. Furthermore, its 40-pin GPIO header enables seamless interfacing with external sensors and hardware components, making it adaptable for a wide spectrum of embedded projects and IoT applications. Operating on the NVIDIA JetPack SDK and Ubuntu Linux, the Jetson Nano provides a familiar development environment well supported with libraries, APIs, and tools tailored for AI development.

Despite its robust performance capabilities, the Jetson Nano remains remarkably power-efficient, rendering it suitable for battery-powered and embedded applications where power consumption is a concern. Its compact form factor coupled with versatile features has cemented the Jetson Nano as a preferred choice among hobbyists, researchers, and professionals alike, spanning various domains including AI development, robotics, drones, smart cameras, and more.

We have leveraged the abilities of Jetson Nano in our project, especially its GPU abilities and CUDA platform in order to enhance the efficiency of our project as a result.

3.2.3. ESP32:

The ESP32 is a highly versatile microcontroller with an integrated Wi-Fi module, encapsulating considerable computational power within a minute form factor. At the heart lies a dual-core Tensilica LX6 processor, operating at frequencies of up to 240 MHz, thereby possessing substantial processing capabilities adaptable to a spectrum of tasks. Its integration of Wi-Fi and Bluetooth functionalities positions it as an excellent candidate for the development of IoT products, facilitating seamless connectivity with a vast expanse of online resources and inter-device communication channels.

This small yet formidable device is endowed with an extensive array of peripheral interfaces, encompassing GPIO pins, SPI, I2C, UART, among others, thereby affording a degree of flexibility helpful to interfacing with a diverse array of sensors, displays, and actuators. Distinguished by its conservative power consumption, the ESP32 aligns itself suitably with battery-powered applications, while its resilient performance guarantees unwavering operation, even in environments of rigorous demands.

The ESP32 is equipped with both Flash memory and RAM, essential components for storing program instructions, variables, and other essential data. The Flash memory serves as the non-volatile storage medium, housing the firmware and application code. With its generous Flash memory capacity, often ranging from 4MB to 16MB or more, the ESP32 offers ample space for storing program binaries, configuration data, virtual filesystem, and additional resources, thus accommodating the diverse requirements of embedded applications.

Moreover, the ESP32 is supported by a robust development ecosystem, comprising software development kits (SDKs), libraries, and tools, from

Arduino and Espressif (ESP-IDF framework) which streamline the development process and accelerate time-to-market for IoT projects. Supported programming languages include C, C++, and MicroPython, catering to a diverse audience of developers with varying skill levels and preferences.

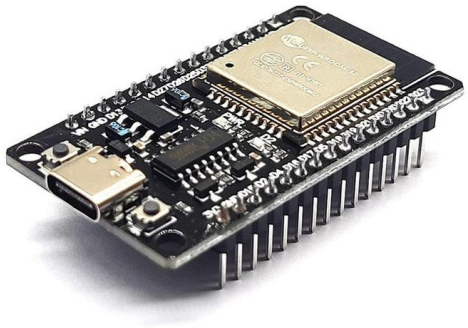

Figure 15. ESP32

3.2.4. BTS7960:

The BTS7960 is a robust and efficient motor driver module designed to control DC motors with high precision and reliability. Equipped with H-bridge circuitry, it supports a wide input voltage range from 5V to 27V, with a maximum continuous current rating of up to 43A. This enables bidirectional control of motor speed and direction, making it suitable for a diverse range of applications, including robotics, automotive projects, and industrial automation. With its compact form factor and integrated protection features against overcurrent, overheating, and short circuits, the BTS7960 ensures both versatility and durability in driving DC motors with optimal performance and safety.

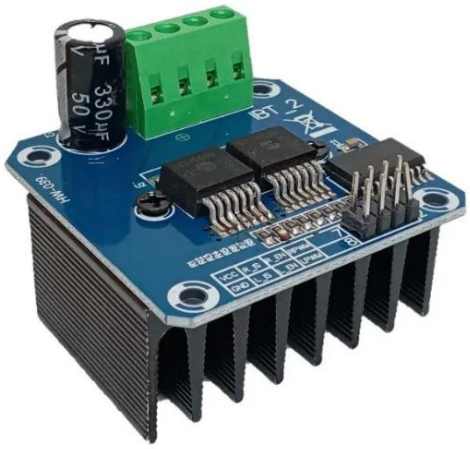

Figure 16. BTS7960

3.2.5. L298N:

The L298N motor driver is a highly versatile and widely used integrated circuit (IC) designed for controlling DC motors and stepper motors. Its dual H-bridge configuration allows independent control of two motors, supporting bidirectional movement and precise control. With a wide operating voltage range of 5V to 35V and a maximum current rating of 1.5A along with compatibility with both TTL and CMOS logic levels, the L298N is adaptable to various power sources and microcontroller systems. Additionally, built-in protection features such as thermal shutdown and overcurrent protection enhance the reliability and durability of the motor driver.

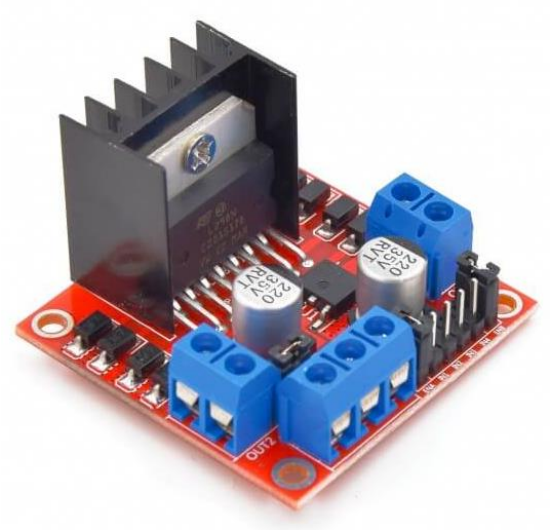

Figure 17: L298N

3.2.6. Rechargeable Sealed Lead Acid Battery:

The 12V 7AH rechargeable sealed lead-acid (SLA) battery is a dependable power source suitable for various applications. Its sealed design ensures safety and reliability by preventing electrolyte leakage, making it ideal for use in portable devices, emergency lighting systems, security setups, and more. With a capacity of 7AH, it can deliver a steady output of 7 amps for one hour or lower currents for extended periods. The rechargeable nature of the battery allows it to be replenished multiple times, reducing waste, and providing a sustainable power solution. Commonly employed in security systems, emergency lighting, UPS setups, portable electronics, and solar energy storage, this SLA battery plays a crucial role in ensuring uninterrupted power supply in our project.

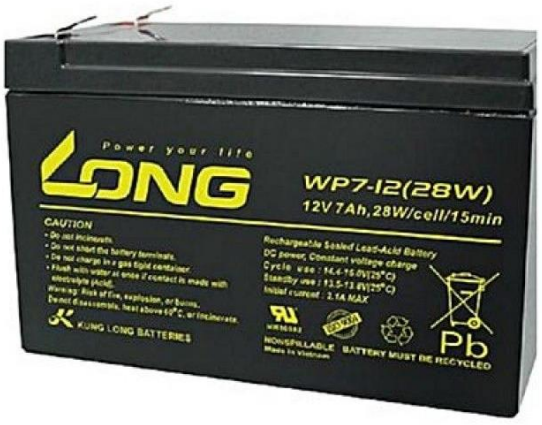

Figure 18: SLA Battery

3.2.7. Mobile Base:

For the mobile base platform that will act as our autonomous vehicles, we have utilized the four-wheel mobile platform that is in the Robotics lab in the Mechatronics Department, NUST. It has a 4-wheel drive system, which is powered by 4 independent motors. There is also a steering mechanism which is controlled using a 5th motor. Each motor has a 12V and 2A stall current rating and is supplied power from a 12 V sealed lead acid battery. The battery has a 7AH rating as well to provide enough current in the scenario that the mobile base is forced to be stationary when moving.

Figure 19: Mobile Platform Utilized in this Project

3.3. Software and Environment

3.3.1. Python:

Python offers several advantages, with scalability being a primary benefit, enabling the handling of large datasets efficiently. It excels in managing and processing data without overwhelming memory resources. Additionally, Python's versatility extends to various domains, including data processing, web development, and scientific computing, due to its adaptability to a wide range of tasks. Furthermore, the abundance of libraries further enhances its utility, particularly in projects involving computer vision tasks, making Python the preferred choice for such endeavors. In our project, we leverage several commonly used libraries to implement code, facilitating seamless execution of tasks. The libraries used in our project are as follows:

1) **OpenCV:** This Python library specializes in computer vision and image processing tasks, offering straightforward functions for tasks such as loading, manipulating, and saving image data. In our project, we

employed it specifically for displaying the semantic classification mask obtained from the model.

2) **NumPy:** This library is geared towards numerical computation, facilitating manipulation of extensive matrices and vectors. NumPy empowers users to execute various mathematical models, including Fourier analysis and random number generation. In our project, we have utilized it in order to manipulate the Ground truth masks and even in some of the data processing that was required in our project from the image obtained.
3) **Matplotlib:** This library serves as a robust tool for generating visualizations and plots within Python. With it, users can effortlessly create diverse visual representations such as line plots, scatter plots, bar charts, and histograms. Additionally, it extends support to 3D plots and visualizations, enhancing the depth and complexity of graphical presentations. We have used it to display and compare the output image with input image in our project.
4) **Hugging Face libraries**: The Hugging Face library, known as "Transformers," is a leading open-source tool for development of AI models and is built upon TensorFlow and PyTorch. In recent updates, Hugging Face has incorporated vision transformer models, such as ViT (Vision Transformer), DeiT (Data-efficient image Transformer), and DETR (DEtection TRansformer), which are designed for various computer vision tasks including image classification, object detection, and image segmentation. These models leverage transformer architectures, originally developed for NLP tasks, and adapt them to handle computer vision tasks effectively. With Hugging Face's support for these models, users can easily access, fine-tune, and deploy vision transformers for a wide range of computer vision applications, further broadening the library's utility across multiple domains. In our project,

we have used this to implement our model, Segformer, which is a vision based semantic classification model.

5) **PYCUDA:** CUDA (Compute Unified Device Architecture) is a sophisticated platform and programming model created by NVIDIA, which harnesses Graphics Processing Units (GPUs) to accelerate computing tasks. It enables developers to use popular programming languages such as C++, Python, and MATLAB, with parallelism expressed through specific keywords. The CUDA Toolkit comprises libraries, a compiler, and development tools necessary for building GPU-accelerated applications. When employing CUDA, the workload is efficiently divided between the CPU and numerous GPU cores, enabling the swift processing of computationally intensive parts of applications. This parallel processing capability significantly boosts the performance and speed of various computational tasks, making CUDA an indispensable tool in areas such as high-performance computing, scientific simulations, and machine learning. We have used PYCUDA library in our project as it provides the python API bindings for CUDA APIs so that we can accelerate the model's inference speed that we have developed on the Jetson Nano.

3.3.2. Google Colab:

Google Colab, short for Google Colaboratory, is a cloud-based platform provided by Google that enables users to run and execute Python code in a collaborative environment. Built on top of Jupyter Notebooks, Colab offers a range of features including free access to computing resources such as CPUs, GPUs, and TPUs, making it ideal for tasks requiring significant computational power, such as machine learning and data analysis. Users can write and execute code directly in the browser, with the option to import datasets, install libraries, and visualize results seamlessly. Moreover, Colab

facilitates collaboration by allowing users to share notebooks with colleagues or work collaboratively in real-time. With its ease of use, powerful computing resources, and collaborative features, Google Colab has become a popular choice for students, researchers, and professionals seeking a versatile and accessible platform for Python programming and data analysis tasks. In our project, we have used Google Colab in order to train our semantic scene classification models using the GPU provided by Google Colab.

3.3.3. Archiconda3:

Archiconda3 is a distribution of conda specifically tailored for 64-bit ARM architectures. Conda is a package management system and environment management system that simplifies the installation and management of software packages across various programming languages, including Python and R. Archiconda3, on the other hand, is a popular free and open-source distribution that includes Conda along with a curated collection of packages and libraries for scientific computing, data science, machine learning, and large-scale data processing. Archiconda3 aims to streamline package management and deployment, making it easier for users to set up and work with complex software environments for their data analytics and scientific computing projects. Archiconda3 is being utilized in our project on Jetson Nano to set up different python environments and avoid clashes between different libraries. It also allows the use of different python versions in different environments which differ from the installed system-wide version.

3.3.4. PyTorch:

PyTorch is an open-source deep learning framework developed by Facebook's AI Research lab (FAIR). Renowned for its flexibility, simplicity, and dynamic computational graph construction, PyTorch is one

of the most popular choices among researchers and developers in the field of artificial intelligence.

At the center of PyTorch lies its dynamic computation graph mechanism, which enables users to define and manipulate computational graphs on-the-fly during runtime. Unlike static graph frameworks like TensorFlow, where the computational graph must be defined before execution, PyTorch allows for dynamic graph construction, making it more intuitive and conducive to experimentation. This dynamic nature facilitates faster prototyping and debugging, as users can easily inspect and modify the computation graph as needed.

PyTorch's core philosophy centers on delivering a seamless and Python-centric experience for deep learning development. Its API is designed to be intuitive and expressive, leveraging Python's power and flexibility to create a familiar environment for developers. This Pythonic approach extends to its debugging tools, visualization libraries, and integration with popular Python libraries like NumPy, SciPy, and Matplotlib, ensuring smooth interoperability and enhancing productivity.

Furthermore, PyTorch offers extensive support for GPU acceleration through its torch.cuda module. Users can seamlessly move tensors and models between CPU and GPU devices, leveraging the computational power of NVIDIA GPUs for accelerated training and inference. This GPU acceleration, combined with PyTorch’ s dynamic computation graph and automatic differentiation capabilities, enables efficient and scalable deep learning workflows.

3.3.5. ONNX Framework:

The Open Neural Network Exchange (ONNX) framework is a cutting-edge, open-source ecosystem designed to facilitate interoperability and portability

across deep learning frameworks. Its core function revolves around providing a standardized format for representing neural network models, enabling seamless exchange between frameworks like PyTorch, TensorFlow, and MXNet. This standardized format ensures that models trained in one framework can be exported to ONNX format and seamlessly imported into another framework without requiring extensive reimplementation. This interoperability enables users and developers to leverage the strengths of different frameworks for their specific tasks, without being constrained by a single framework's limitations.

At the core of ONNX lies its unified computational graph representation, which comprehensively captures the structure and parameters of deep learning models in a vendor-neutral format. This representation includes essential information about the network architecture, layer configurations, and connections between layers, along with numerical values of model parameters such as weights and biases. By standardizing the representation of models, ONNX facilitates effortless exchange and execution of models across different runtime environments, ensuring consistency and compatibility throughout the model deployment process.

Moreover, ONNX offers unparalleled deployment flexibility, allowing deep learning models to be deployed across a wide spectrum of platforms and devices. Especially for edge devices, ONNX-compatible models can be optimized and compiled for specific hardware targets using specialized inference engines. This optimization process ensures efficient execution and maximum performance, even in resource-constrained environments. This adaptability makes ONNX particularly well-suited for real-world applications where performance, scalability, and deployment environments may vary.

3.3.6. TensorRT:

TensorRT, developed by NVIDIA, is a sophisticated deep learning optimization and inference engine designed to leverage the full power of NVIDIA GPUs for deploying deep learning models in production environments. At its core, TensorRT focuses on accelerating the inference phase of deep learning models, enabling them to run efficiently and with high performance on NVIDIA GPUs.

TensorRT is a layer-agnostic library that delivers high performance, low latency inference through the optimization of neural network models. The techniques used to optimize neural network models are applied to the computational graph of a deep learning model to streamline its execution and maximize GPU utilization. An example of such tactic is layer fusion, which combines multiple operations into a single optimized kernel, reducing overhead and improving computational efficiency. It also performs precision calibration to quantize model weights and activations to lower precision formats like INT8, which can significantly reduce memory footprint and improve inference speed, especially on embedded platforms and edge devices.

Furthermore, TensorRT incorporates kernel auto-tuning, a process where the optimal kernel configurations are automatically selected based on the characteristics of the underlying hardware. This ensures that the deep learning model is efficiently executed on the specific GPU architecture it is deployed on, leading to optimal performance. Additionally, TensorRT dynamically manages tensor memory during inference, minimizing memory allocations and deallocations to reduce overhead and improve throughput.

Another notable aspect of TensorRT is its support for various precision modes, including FP32 (single-precision floating point), FP16 (half-precision floating point), and INT8 (8-bit integer). This flexibility allows

developers to choose the appropriate precision for their application, balancing accuracy with performance and memory requirements. TensorRT seamlessly integrates with the popular deep learning frameworks, TensorFlow and ONNX, allowing developers to optimize and deploy models trained in these frameworks with ease. It provides both Python and C++ APIs for integration into existing workflows and applications, making it accessible to a wide range of developers.

3.3.7. Espressif IoT Development Framework:

The Espressif IoT Development Framework (ESP-IDF) serves as the backbone of software development for Espressif's ESP32 and ESP32-S series of microcontrollers, offering a comprehensive suite of tools, libraries, and APIs tailored for embedded systems and IoT applications. At its core, ESP-IDF provides developers with a robust and flexible SDK that streamlines the entire firmware development process, from initial prototyping to production deployment. This SDK encompasses a wide range of functionalities, including device drivers for peripherals such as GPIO, UART, I2C, SPI, and ADC, as well as support for networking protocols like Wi-Fi, Bluetooth, and TCP/IP. Moreover, ESP-IDF includes system services for power management, task scheduling, memory management, and error handling, ensuring the reliability and efficiency of embedded applications.

Built upon the FreeRTOS real-time operating system, ESP-IDF offers a solid foundation for building responsive and scalable IoT applications. FreeRTOS provides preemptive multitasking capabilities, allowing developers to create multiple tasks that run concurrently and efficiently manage system resources. This enables developers to implement complex functionalities and handle asynchronous events with ease, making ESP-IDF well-suited for a wide range of IoT use cases, from simple sensor nodes to

sophisticated connected devices.

3.4. Training the AI Models

3.4.1. Segformer:

Segformer[40] represents a groundbreaking approach to semantic segmentation in computer vision by integrating transformers, renowned for their success in natural language processing, into the domain of image analysis. The architecture uniquely blends convolutional and transformer networks, marking a significant departure from conventional CNN-based models. Images are tokenized into non-overlapping patches, and positional encoding is applied to maintain spatial information. The transformer encoder captures global context and long-range dependencies, while a decoder network upscales the features to produce the final pixel-wise segmentation map, preserving local feature extraction. Leveraging the self-attention mechanism intrinsic to transformers, Segformer excels in understanding relationships between distant image tokens, enabling it to grasp the broader context and make informed segmentation decisions. This innovative model offers superior capabilities in capturing global context, ensuring enhanced segmentation accuracy, and accommodating diverse image resolutions with ease. Segformer' s adaptability and performance make it an attractive solution for a variety of computer vision applications, particularly those demanding precise semantic segmentation in complex and dynamic environments[40].

The fundamental principle underlying Segformer' s operation lies in its treatment of the input image as a sequence of fixed-size patches, akin to breaking down a sentence into individual words. Each patch is then subjected to an embedding process, transforming it into a lower-dimensional vector representation. This initial transformation enables the model to process the image in a more structured and manageable manner,

paving the way for subsequent analysis. However, what truly sets Segformer apart is its innovative utilization of self-attention mechanisms, a hallmark of transformer architectures. Self-attention mechanisms empower the model to discern intricate spatial dependencies and contextual relationships across the entire image, allowing it to capture both local and global context effectively.

A pivotal aspect of Segformer' s design is its incorporation of multiple stacked transformer encoder layers. These layers operate sequentially, each processing the embedded patches independently through self-attention mechanisms, followed by feedforward neural network layers. This hierarchical structure facilitates the extraction of increasingly abstract features, thereby enabling the model to comprehend complex spatial relationships within the image. Moreover, Segformer introduces the concept of cross-encoder attention, which enables the model to capture relationships between patches across different scales or levels of abstraction. This adaptive mechanism plays a crucial role in enhancing the model's ability to handle objects of varying sizes and complexities within the image.

During the training phase, Segformer learns to optimize its parameters by minimizing a predefined loss function, typically cross-entropy loss, against ground truth annotations. This process involves iteratively adjusting the model's parameters to improve its performance on the given task. Inference, on the other hand, entails passing an input image through the trained Segformer model, which outputs a probability distribution over semantic classes for each pixel. The final segmentation mask is then generated by assigning the class with the highest probability to each pixel.

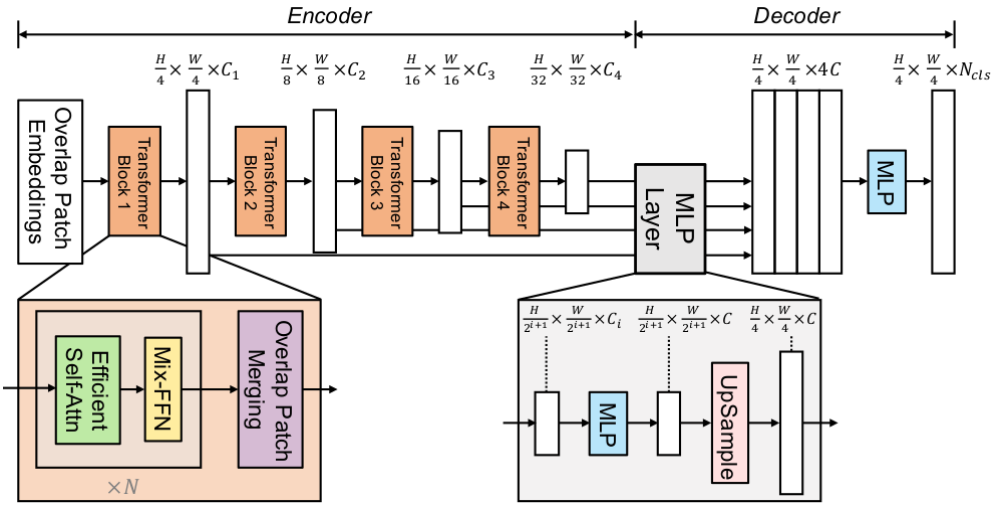

Figure 20. Segformer Architecture

In our case, we have utilized the model encoder segformer-b0, which is the lightest among all of its versions with 3.1 million parameters and we have chosen the input size of the image to be of resolution 512x512. These requirements are fulfilled just enough for Jetson Nano to run.

3.4.2. SegNet:

SegNet stands out for its efficiency, designed for real-time semantic segmentation. Its symmetric architecture includes an encoder and decoder. The encoder extracts features, and the decoder upscales them for pixel-wise predictions. The model emphasizes the importance of preserving spatial information while remaining computationally efficient[41].

The encoder module of SegNet utilizes a series of convolutional layers to progressively extract hierarchical features from the input image, capturing intricate spatial information at multiple scales. These features are then passed through a pooling layer, which down-samples the spatial dimensions while preserving essential semantic information.

Subsequently, the decoder module employs up-sampling operations to recover the spatial resolution of the feature maps generated by the encoder.

By iteratively refining and reconstructing the feature maps, the decoder produces dense predictions for each pixel in the input image.

One of SegNet's distinguishing characteristics is its incorporation of skip connections between corresponding encoder and decoder layers. These skip connections facilitate the propagation of fine-grained spatial details from the encoder to the decoder, enhancing the model's ability to capture precise object boundaries and semantic information.

We have also selected this model to test the results of Segformer model against an older, commonly used and widely supported semantic segmentation model.

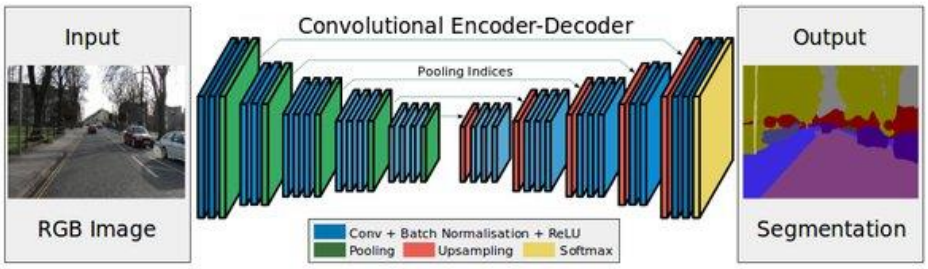

Figure 21: SegNet’s architecture

The resolution of input images was selected to be 1024x512 pixels for SegNet model.

3.4.3. DeepLabV3:

DeepLabV3 is a state-of-the-art deep learning model for semantic image segmentation, developed by the Google Research team. Building upon its predecessors, DeepLabV1 and DeepLabV2, DeepLabV3 incorporates several key innovations to achieve more accurate and efficient segmentation results. One of its notable features is the employment of atrous convolution (also known as dilated convolution), which allows the model to capture multi-scale contextual information without increasing the number of parameters.

DeepLabV3 also integrates atrous spatial pyramid pooling (ASPP), a technique that captures contextual information at multiple scales using parallel atrous convolutions with different dilation rates. Additionally, DeepLabV3 utilizes deep supervision, where intermediate feature maps are used to generate segmentation predictions, facilitating better gradient flow during training, and improving segmentation performance. With these advancements, DeepLabV3 has demonstrated superior performance in various semantic segmentation benchmarks, making it a popular choice for tasks such as object detection, scene parsing, and image segmentation.

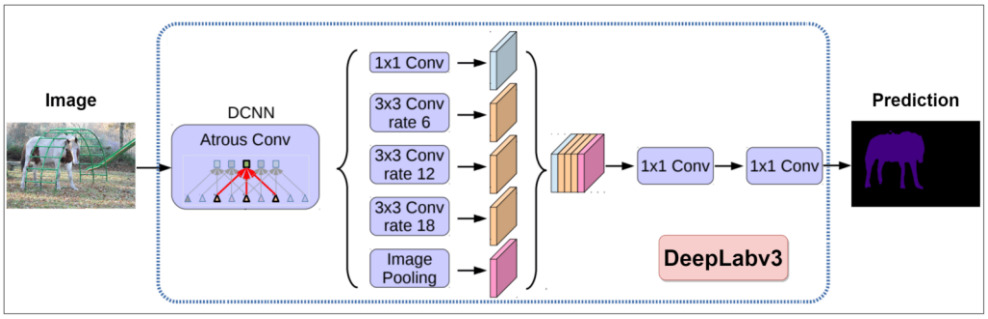

Figure 22: DeepLabV3 Architecture

3.4.4. UNET++:

UNET++ is a semantic segmentation architecture, particularly suited for biomedical image analysis. Its design innovations, including nested skip connections and a DenseNet-inspired encoder, enable the model to capture both local and global context information efficiently. The nested skip connections facilitate the integration of features at multiple scales, while the DenseNet-inspired encoder promotes feature reuse, contributing to more discriminative representations. Additionally, UNET++ incorporates attention gates, allowing the model to selectively focus on informative regions of the input image, further improving segmentation accuracy. These enhancements collectively elevate UNET++ to achieve state-of-the-art performance in biomedical image segmentation tasks, facilitating more accurate and precise analysis for applications such as medical diagnosis and

treatment planning.

The impact of UNET++ extends beyond biomedical imaging, with its architecture demonstrating efficacy in various semantic segmentation applications. By leveraging nested skip connections and attention gates, UNET++ effectively addresses challenges related to capturing spatial context and handling class imbalance. Its versatility makes it applicable to diverse domains, including remote sensing, autonomous driving, and scene parsing. Moreover, UNET++'s modular design and efficient architecture render it suitable for deployment on resource-constrained devices, opening avenues for real-time applications in embedded systems and edge computing environments.

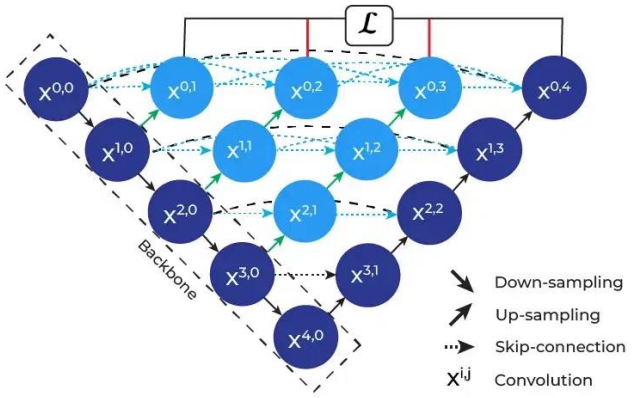

Figure 23: UNET++ Architecture

3.4.5. Cityscapes Dataset:

The Cityscapes dataset stands as a cornerstone resource in the domain of computer vision, particularly in the field of semantic understanding and scene parsing. Comprising high-quality urban street scenes captured across various cities in Germany, Cityscapes offers a rich and diverse collection of images meticulously annotated for semantic segmentation tasks. Its comprehensive nature and meticulously labeled data make it an invaluable asset for training and evaluating state-of-the-art algorithms in semantic

understanding, including tasks such as object detection, instance segmentation, and image classification.

One of the distinguishing features of the Cityscapes dataset is its focus on urban environments, providing a realistic representation of the challenges encountered in real-world scenarios. The dataset encompasses a wide array of scenes, ranging from bustling city streets and intersections to quiet residential neighborhoods, each presenting unique visual complexities and intricacies. This diversity enables researchers and practitioners to develop and test algorithms capable of handling a broad spectrum of urban environments, thus facilitating more robust and generalizable solutions.

In terms of scale, the Cityscapes dataset boasts an extensive collection of high-resolution images, with each image typically containing multiple objects and instances of interest. Furthermore, the dataset offers fine-grained pixel-level annotations for semantic segmentation, wherein each pixel is assigned a specific class label corresponding to objects or regions within the scene. This level of granularity enables precise delineation and understanding of urban scenes, empowering algorithms to accurately identify and classify various elements such as cars, pedestrians, road markings, buildings, and vegetation.

Moreover, Cityscapes provides a hierarchical annotation scheme encompassing a diverse range of semantic classes, thereby capturing the rich semantic structure inherent in urban environments. This hierarchical organization facilitates more nuanced and detailed analysis of scenes, allowing algorithms to distinguish between different types of objects and entities with varying degrees of specificity. Such detailed annotations are instrumental in advancing the state-of-the-art in semantic segmentation and related tasks, enabling researchers to push the boundaries of what is achievable in computer vision.

In addition to its vast collection of images and annotations, the Cityscapes dataset also includes a comprehensive set of evaluation metrics and benchmarks for assessing the performance of algorithms. These metrics encompass various aspects of semantic understanding, including pixel-level accuracy, class-wise segmentation accuracy, and instance-level segmentation metrics. By providing standardized evaluation protocols, Cityscapes facilitates fair comparisons between different algorithms and methodologies, fostering a collaborative and competitive research environment within the computer vision community.

Figure 24: An Image and its Segmentation Mask from Cityscape Dataset

It is to be noted that we changed the dataset to reduce the number of classes by merging many of the similar classes like for example merging the bicycle and motorbike classes and another example is merging the bus class with the truck class. The classes to merge were selected based on their level of occurrence in the dataset and the possibility of appearing in a real-life scenario. This merging results in an increase in prediction accuracy because

theoretically the model can't learn features that are rare so merging rare identical classes will result in the prediction accuracy of both classes adding up. By merging the classes, we reduced the number of classes from 19 to 11.

3.4.6. Training AI Models on Google Colab:

To train the models at a fast rate, we chose to train them on google colab. As stated earlier in this chapter, google colab offers the privileges of NVIDIA T4 GPUs to users for multiple different tasks for a limited time per day. We will use it to accelerate the training and evaluation time of the models.

Before we can start training the model, we have to upload the modified Cityscape dataset to colab for access during training. One method is to upload the dataset to a google drive and then mount the drive in the colab to access the pictures in the dataset. The downside of this method is the free storage limit on google drive and the supported upload speed is extremely slow. The other method is to upload the dataset to Hugging Face. With no limit on size of dataset and a fast upload speed along with availability of Hugging Face APIs for manipulating the dataset, Hugging Face is the best place to upload and store your own datasets as well as explore community created datasets for quick use.

After the dataset has been uploaded, we wrote a python script using PyTorch framework and transformers library form Hugging Face to train and evaluate the model. To train the Segformer model, we are utilizing the power of transfer learning to train the model. It was initially trained on ade20k dataset which is a dataset consisting totally of 150 semantic categories, which includes classes like sky, road, grass, and other discrete objects. After the desired accuracy and miou were obtained by completing multiple epochs, both the models' weights were downloaded from colab to

use for inference.

3.5. Optimization

3.5.1. Conversion to ONNX Framework:

Since the model was trained using PyTorch framework, its weights will be in .pth format (PyTorch's supported format). To optimize the inference speed of the model, we need to use TensorRT APIs. As stated above, TensorRT only supports the frameworks TensorFlow (TF-TRT) and ONNX. Hence conversion to ONNX framework of the trained PyTorch model weights is inevitable. Here Hugging Face again comes to the rescue by providing a library along with its CLI (Command Line Interface) tools called "optimum". Using optimum with the CLI has made it possible to convert PyTorch models to ONNX with extreme speed and simplicity.

3.5.2. TensorRT Inference Engine:

TensorRT provides C++ and Python APIs to create an inference engine from ONNX framework model weights. NVIDIA also provides a command line wrapper tool "trtexec" for TensorRT APIs, which once natively compiled, can be used to create the inference engine from the command line.

When creating an inference engine, TensorRT performs multiple optimizations and provides different levels of quantization to reduce model size and increase inference speed by decreasing precision of model's weights. It performs multiple different tactics and times the performance of each one and selects the one with the best performance. Successful completion of the process of creating an inference engine will create a ".engine" file which can then be used along with TensorRT Runtime APIs to perform model inference.

3.6. IoT Control for Mobile Base

3.6.1. Reason:

The main reason for setting up an IoT control for the mobile base is to provide a failsafe control switch over to a remote user. In case the autonomous drive control system working based on the semantic scene classification (beyond the scope of this project) fails, a remote user can take over control of the vehicle to prevent a road-side accident.

3.6.2. Implementing Mobile Base Control Scheme:

Since the mobile base consists of 5 DC motors, it can be powered using an SLA battery. Motor driver modules such as BTS7960 which have higher current rating can be used to drive the motors controlling each tire of the mobile base because of the likely possibility of high torque required due to obstacles in path of motion of base. However, motor driver modules with lower current rating such as the L298N can be used to control the steering motor since it doesn't need to face higher torques and there is less chance of stalling. Both modules can then be interfaced to an ESP32 or any other microcontroller that supports 5V or 3.3V logic. Then said microcontroller can receive inputs from an IoT network to control the motors of the base.

3.6.3. Setting up the IoT Network:

ESP32 microcontroller has built-in Wi-Fi and Bluetooth capabilities so the ESP32 can serve as a host server for an IoT network. The ESP-IDF provides an extensive number of APIs to configure the implementation of a server on the ESP32. Along with the configuration of the back end of the server, a front-end, like a website, can also be stored in the ESP32 to provide a simple GUI (Graphical User Interface) dashboard to control the mobile base. A front end can be implemented using the popular languages JavaScript, Vue,

TypeScript etc. We will be using JavaScript to create a front end due to simplicity of the language and prior knowledge. Moreover, we will also be using HTML and CSS to aid with the creating of a front-end GUI.

To use JavaScript for website hosting, we need to implement the WebSocket protocol to handle communication between client and server. WebSockets on ESP32 enable bidirectional communication between an ESP32 microcontroller and a web server, facilitating real-time data exchange over a network connection. Using the WebSocket protocol, ESP32 devices can establish persistent and low-latency connections with web servers, allowing for efficient and responsive communication in various IoT applications. By establishing WebSocket connections, ESP32 devices can receive and transmit data streams, sensor readings, commands, and notifications in real time, enabling seamless interaction with web-based applications, mobile devices, and other IoT endpoints.

Access Point (AP) mode, also known as SoftAP (Software Access Point) mode, enables a Wi-Fi-enabled device to act as a standalone Wi-Fi network access point. In this mode, the device creates its own Wi-Fi network, allowing other devices to connect to it just like they would to a traditional Wi-Fi router. AP mode is particularly useful in scenarios where a standalone network needs to be established. By enabling AP mode, devices can provide network connectivity to other devices, enabling them to communicate, share data, and access the internet without the need for a pre-existing Wi-Fi infrastructure. Hence, the ESP32 can be configured into AP mode to allow clients to directly connect to the ESP32 to directly communicate to the server.

Once connected to the ESP32's network, the server needs to send the JavaScript, CSS, and HTML scripts to the client, so the front end is visible in the client's web browser. To send the scripts, first the ESP32 needs to

store them in flash memory. SPIFFS (Serial Peripheral Interface Flash File System) serves as a lightweight and efficient file system specifically optimized for use with SPI flash memory. ESP32's SPIFFS allows storage and management of files directly on the flash memory chip integrated into the ESP32 module.

3.7. Summary

In this chapter, we have discussed the different tools that would be needed to implement this project and the different methodologies to use those tools, for example training the neural networks and setting up the IoT Network. In the next chapter, we will show the results of adopting these methodologies long with the details needed for the hardware integration.

Chapter 4 – RESULTS

4.1. Overview:

Chapter 4 presents the results of the semantic scene classification models and the IoT network implemented for mobile base control. The chapter begins with the test results on test data of all 4 neural networks mentioned in chapter 3. Then after picking the two models, Segformer and SegNet, it depicts their performance on KITTI and real-life benchmarks. Lastly, it details the IoT network's performance and precision in controlling the mobile base along with the designing and manufacturing of the camera mount being used in this project.

4.2. Data Collection and Analysis:

To evaluate the precision and accuracy of our semantic scene classification models, we performed experiments in two different settings. Firstly, we tested the AI models using datasets sourced from KITTI, which includes real-life road scenarios from rural landscapes and highways. Following this, we assessed the performance of our algorithm using real-world data, gathered from the Raspberry PiCam V2.1, positioned near our university.

4.3. Testing the Semantic Scene Classification Models:

After training the models on google colab, the model parameters on test data are stated in the table below:

Table 3: Trained Model Parameters

Name	Segformer	DeepLabV3	UNET++	SegNet

Mean Iou	0.76	0.69	0.73	0.58

Among these models, Segformer had the best accuracy and was selected to be deployed on the Jetson Nano. The training loss curve for the Segformer model is as follows:

Figure 25: Training Loss Curve for Segformer Model

SegNet was also selected because it had the lightest architecture compared to DeepLabV3 and UNET++, as well as to demonstrate a range between the best mean Iou. Here is an inference on an image from Cityscape's test dataset (not used to train the model) long with its ground truth:

Figure 26: Image from Cityscape test dataset

Figure 27: Segmented Image from Segformer Model (on PC)

Figure 28: Ground Truth Mask for Figure 26

The Segformer model without any optimization was also evaluated on KITTI dataset. The results are as follows:

Figure 29: Image (1) from KITTI Dataset

Figure 30: Segmented Image (1)

Figure 31: Image (2) from KITTI Dataset

Figure 32: Segmented Image (2)

The model was also evaluated on real-life data as well. The results are as follows:

Figure 33: An Image from Islamabad Road

Figure 34: Segmented Image of Figure 29

The model was then ported onto the Jetson Nano and optimized. The optimization was done by quantizing the model weights to INT8 and FP16. This resulted in a slight decrease in prediction accuracy but a speed up in the inference process. The

FP16 quantized model was then tested using real-time video obtained by interfacing the PiCam. The results are as follows:

Figure 35: Legend for the Segmented Images below

Figure 36: Image (1) Captured in Our University

Figure 37: Segmented Image of Figure 32

Figure 38: Image (2) Captured in Our University

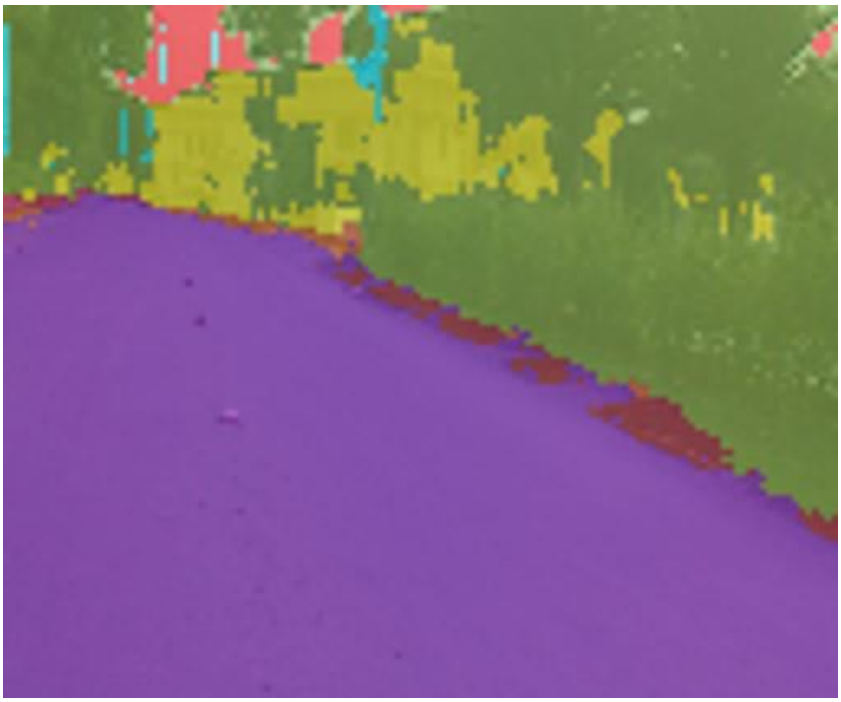

Figure 39: Segmented Image of Figure 33

Figure 40: Image (3) Captured in Our University

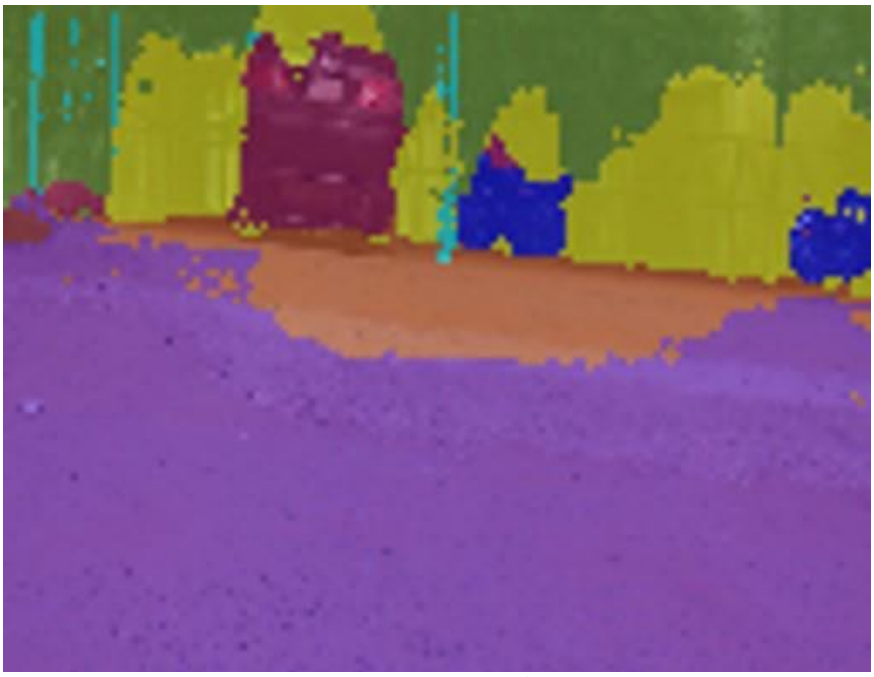

Figure 41: Segmented Image of Figure 35

The frames per second (FPS) for each stage of the Segformer model is listed in the table:

Table 4: FPS of Segformer model

	Segformer (not optimized)	Segformer (FP16 quantization)	Segformer (INT8 quantization)
Mean FPS	0.98	1.71	1.25

Segformer with FP16 quantization is faster than the INT8 quantized version because the CUDA kernels on 64-bit Arm architecture are optimized for floating point operations. Hence performing INT8 calculations will be slower on the Jetson Nano even though the INT8 model size is much smaller than the FP16 one. Needless to say, the original model with FP32 precision is the slowest of them all because of the greater memory footprint and large number of bits for calculation.

Following the conclusion from the previous paragraph, SegNet was also quantized to FP16 for the best speed performance. The FPS obtained had a mean average of 4.0 which is to be expected from a smaller model. The results on the Jetson Nano are as follows:

Figure 42: SegNet Inference on Jetson Nano

It can be seen that the output of the SegNet is less accurate than Segformer's by a large margin. The benefit of each model is either extremely good accuracy in Segformer's case and a fast real-time feed in SegNet's case.

4.4. IoT Network:

The IoT network was implemented as discussed in chapter 3. The GUI of the webpage is as follows:

Figure 43: IoT Webpage GUI

The webpage can be accessed by connecting to the ESP32's Wi-Fi. It cannot be accessed using any other Wi-Fi because the website is a locally hosted one as mentioned before. The website has 4 buttons each making the base perform the motion it has stated. Pressing a button once will initiate the motion and pressing it again once more will stop it. A combination of forward/reverse and left/right buttons can be pressed to steer the base while it is moving back and forth.

This website was of the highest complexity that could possibly be hosted on the ESP32 because of storage constraints. A more complex website would require more code memory to host itself, which isn't possible because the flash memory of ESP32-DevKitC is 4 MB and as such the partition table only allows 1 MB of storge for SPIFFS.

4.5. Camera Mount:

To hold the PiCam in the correct location precisely, we need to design and fabricate a camera mount. The 2D drawings for the camera mount are as follows:

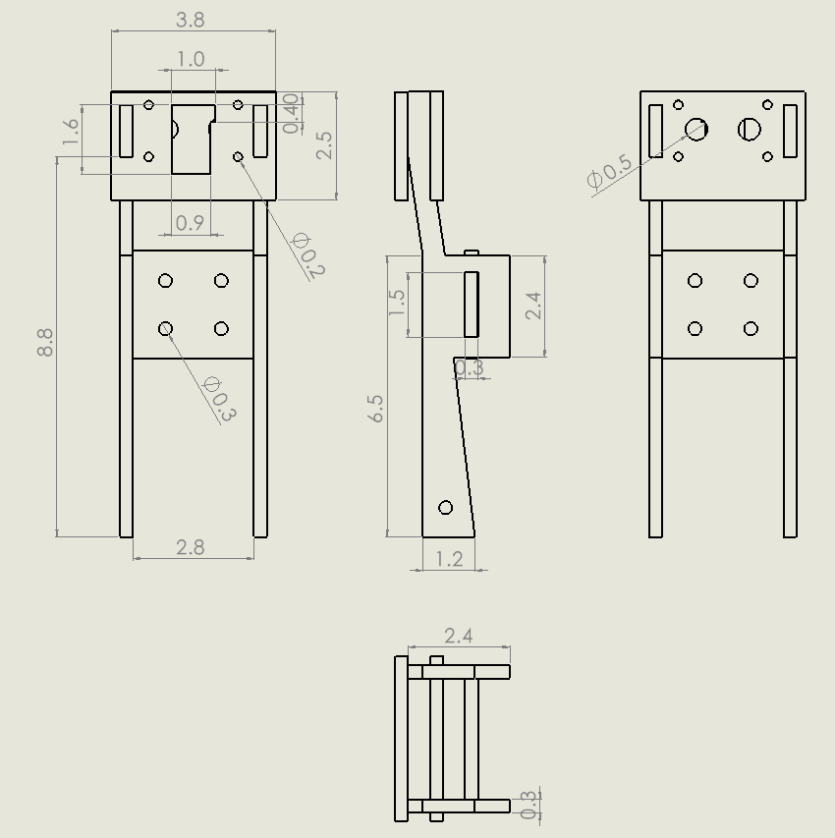

Figure 44: Camera Mount Design

This design was used to manufacture the camera mount using a 3D printer. By importing this CAD model design into a 3D printing software, we selected the suitable parameters, and a desired printing file is generated. Later this file is sent to a 3D printer using additive manufacturing to create the physical object layer by layer. The manufactured camera mount looks like the follows:

Figure 45: Manufactured Camera Mount

4.6. Hardware Integration:

The camera mount was fixed at the front of the base in a position to acquire a live video feed at a considerable height simulating the height of a road vehicle. The Jetson Nano is interfaced with the camera using one of its CSI connectors. Furthermore, an LCD with an HDMI interface is also connected with the Jetson Nano to display the segmented scene. The ESP32 microcontroller is interfaced with a BTS7960 and L298N motor driver modules to control the base's motion. The battery is connected to the motor driver modules as well to power the motors. A power bank was used to power the ESP32 and the Jetson Nano.

Figure 46: Mobile Base Integrated with Sensors

4.7. Summary:

As observed in the results displayed in the fourth chapter, Segformer is the best neural network in terms of speed and accuracy combined out of all the neural networks tested in this project. Its lightweight architecture with only 3.1 million parameters allows for real-time inference while still retaining the advantages of a vision transformer in terms of accuracy. The IoT network implemented using ESP32 provides the flexible control in case of any critical situation. In the next chapter, we will conclude the finding of our project and provide further improvements for future research.

Chapter 5 – CONCLUSIONS

5.1. Summary of Achievements:

We were able to port the Segformer-b0 model onto the Jetson Nano and were able to run the model in real time at 1.71 fps with great accuracy only with an input resolution of 512x512 pixels. Also, control method was established over IoT using an ESP32 for a mobile base.

Undertaking this final year project, “Semantic Scene Classification using AI for Autonomous Vehicles”, has offered several benefits. Firstly, it allows for the exploration and understanding of the application of neural networks in the field of autonomous vehicles and scene perception. It taught the ways for developing and analyzing multiple neural networks with different architectures for an autonomous vehicle. Secondly, the procedure of porting an AI algorithm for edge devices to later run inference locally was learned by implementing this project. Multiple different techniques for optimization of deep learning models for deploying on edge devices were studied and applied to analyze the amount of performance gained and lost in different sectors. Thirdly, the development of an IoT network for the precise control of the base and the intricacies of different network protocols were also underlined and worked upon in this project. Lastly, by addressing these challenges, this project contributes to the broader research community's understanding of scene classification systems, potentially leading to advancements in autonomous driving technology and improved safety on the roads. This project allowed for the development of a platform on which more sensing modalities can be added which would lead to further investigation in the domain of Autonomous Vehicle navigation in the department of Mechatronics Engineering. Overall, undertaking this final year project offers invaluable knowledge, skill development, and a chance to make a meaningful contribution to the field of autonomous vehicles and edge AI.

5.2. Future Improvements:

- Use an SBC with better specifications to run the AI models at a higher FPS. Even though Jetson Nano with 128 CUDA cores is good enough to run most deep learning algorithms, it still has its limitations that come with an entry-level product. Using an SBC with more CUDA cores, better GPU architecture or with support of INT8 operations would significantly speed up the real time system. Some of the SBCs possessing these features are the Jetson TX2, Jetson Xavier NX and Jetson AGX Xavier lineup.
- Use a microcontroller integrated with a larger flash memory. As explained in the previous chapter, the 4MB integrated flash memory limits the allowable complexity of the website that the microcontroller can host. To implement a more precise control system over IoT, for example a joystick controlling the mobile base or real time speed control in each direction of motion of the mobile base. Some devboards of ESP32S3 series as well as ESP32C6 series can have 8MBs of flash memory or more.
- Train Segformer model with a higher input resolution to classify objects far in a picture. An input resolution of 512x512 pixels means that classes that are far away and smaller in the image cannot be classified accurately because further down sampling in the model's architecture removes those features. Training the model with a higher input resolution should help with this problem.
- Integrate depth perception for complete perception of scene. This is the next possible step for this project to completely develop the perception part of an autonomous vehicle. Fusion of both high level and low-level perception should allow for development of path planning, object detection and control system algorithms progressing towards a fully autonomous vehicle.

REFERENCES

[1] T. P. published, “How do self-driving cars work? Everything you need to know,” Tom’s Guide, Sep. 11, 2022. https://www.tomsguide.com/reference/how-do-self-driving-cars-workeverything-you-need-to-know (accessed Oct. 28, 2023).

[2] “Autonomous Cars: Levels of Autonomous Driving Explained,” AckoDrive, May 11, 2022. https://ackodrive.com/car-guide/autonomous-cars-and-levelsof-autonomous-driving/ (accessed Oct. 28, 2023).

[3] “Semantic Segmentation for Autonomous Driving | SpringerLink.” https://link.springer.com/chapter/10.1007/978-981-19-9304-6_61 (accessed Oct. 28, 2023).

[4] P. -Y. Hsiao, C. -W. Yeh, S. -S. Huang and L. -C. Fu, "A Portable Vision-Based Real-Time Lane Departure Warning System: Day and Night," in IEEE Transactions on Vehicular Technology, vol. 58, no. 4, pp. 2089-2094, May 2009, doi: 10.1109/TVT.2008.2006618.

[5] S. Suchitra Sathyanarayana, R. K. Satzoda and T. Srikanthan, "Exploiting Inherent Parallelisms for Accelerating Linear Hough Transform," in IEEE Transactions on Image Processing, vol. 18, no. 10, pp. 2255-2264, Oct. 2009, doi: 10.1109/TIP.2009.2026680.

[6] Akbari Sekehravani, Ehsan & Babulak, Eduard & Masoodi, Mehdi. (2020). Implementing canny edge detection algorithm for noisy image. Bulletin of Electrical Engineering and Informatics. 9. 1404-1410. 10.11591/eei.v9i4.1837.

[7] Noureddine, Chibani & Sebbak, Faouzi & Cherifi, Walid & Belmessous, Khadidja. (2022). Road anomaly detection using a dynamic sliding window technique. Neural Computing and Applications. 34. 1-19. 10.1007/s00521-022-07436-6.

[8] Hongzhi Zhou, Gan Yu, Research on pedestrian detection technology based on the SVM classifier trained by HOG and LTP features, Future Generation Computer Systems, Volume 125, 2021, Pages 604-615, ISSN 0167-739X, https://doi.org/10.1016/j.future.2021.06.016.

[9] B. Ashwini, B. N. Yuvaraju, A. Y. Pai and B. Aditya Baliga, "Real Time Detection and Classification of Vehicles and Pedestrians Using Haar Cascade Classifier with Background

Subtraction," 2017 2nd International Conference on Computational Systems and Information Technology for Sustainable Solution (CSITSS), Bengaluru, India, 2017, pp. 1-5, doi: 10.1109/CSITSS.2017.8447818.

[10] Utomo, Didi & Ummah, Tri & Sulistyaningrum, Dwi & Setiyono, Budi & Soetrisno, & arry sanjoyo, Bandng. (2020). Vehicle detection using histogram of oriented gradients and real adaboost. Journal of Physics: Conference Series. 1490. 012001. 10.1088/1742-6596/1490/1/012001.

[11] Zhuang, Xiaobin & Kang, Wenxiong & Wu, Qiuxia. (2016). Real-time vehicle detection with foreground-based cascade classifier. IET Image Processing. 10. 10.1049/iet-ipr.2015.0333.

[12] J. Long, E. Shelhamer, and T. Darrell, "Fully Convolutional Networks for Semantic Segmentation." arXiv, Mar. 08, 2015. doi: 10.48550/arXiv.1411.4038.

[13] S. Zheng et al., "Conditional Random Fields as Recurrent Neural Networks," in 2015 IEEE International Conference on Computer Vision (ICCV), Dec. 2015, pp. 1529–1537. doi: 10.1109/ICCV.2015.179.

[14] W. Liu, A. Rabinovich, and A. C. Berg, "ParseNet: Looking Wider to See Better." arXiv, Nov. 19, 2015. doi: 10.48550/arXiv.1506.04579.

[15] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, "Pyramid Scene Parsing Network." arXiv, Apr. 27, 2017. doi: 10.48550/arXiv.1612.01105.

[16] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation." arXiv, May 18, 2015. doi: 10.48550/arXiv.1505.04597.

[17] 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." arXiv, May 11, 2017. doi: 10.48550/arXiv.1606.00915

[18] "ReSeg: A Recurrent Neural Network-Based Model for Semantic Segmentation | IEEE Conference Publication | IEEE Xplore." Accessed: Nov. 03, 2023. [Online]. Available: https://ieeexplore.ieee.org/document/7789550

[19] "The Importance of Sensor Fusion for Autonomous Vehicles," Digital Nuage, Dec. 05, 2021. https://www.digitalnuage.com/the-importance-of-sensorfusion-for-autonomous-vehicles (accessed Oct. 28, 2023).

[20] "Papers with Code - Real-Time Semantic Segmentation." https://paperswithcode.com/task/real-time-semantic-segmentation (accessed Oct. 28, 2023).

[21] M. Tan and Q. V. Le, "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks." arXiv, Sep. 11, 2020. doi: 10.48550/arXiv.1905.11946.

[22] R. Strudel, R. Garcia, I. Laptev, and C. Schmid, "Segmenter: Transformer for Semantic Segmentation." arXiv, Sep. 02, 2021. doi: 10.48550/arXiv.2105.05633.

[23] C. Yu, J. Wang, C. Peng, C. Gao, G. Yu, and N. Sang, "BiSeNet: Bilateral Segmentation Network for Real-time Semantic Segmentation." arXiv, Aug. 02, 2018. doi: 10.48550/arXiv.1808.00897.

[24] A. Berthelier, T. Chateau, S. Duffner, C. Garcia, and C. Blanc, "Deep Model Compression and Architecture Optimization for Embedded Systems: A Survey," J Sign Process Syst, vol. 93, no. 8, pp. 863–878, Aug. 2021, doi: 10.1007/s11265-020-01596-1.

[25] Choi, Y., El-Khamy, M., Lee, J., "Towards the Limit of Network Quantization" ICLR, 2017, doi: https://doi.org/10.48550/arXiv.1612.01543

[26] "Single board computers for machine earning & AI | Arrow.com." Accessed: Nov. 03, 2023. [Online]. Available: https://www.arrow.com/en/research-and-events/articles/the-future-of-single-board-computers-and-artificial-intelligence

[27] "FPGAs & 3D ICs," AMD. Accessed: Nov. 05, 2023. [Online]. Available: https://www.xilinx.com/products/silicon-devices/fpga.html

[28] "Intel® FPGAs - Intel® Arria® 10 FPGAs." Accessed: Nov. 03, 2023. [Online]. Available: https://www.intel.com/content/www/us/en/products/details/fpga/arria/10.html

[29] "Edge TPU - Run Inference at the Edge," Google Cloud. Accessed: Nov. 03, 2023. [Online]. Available: https://cloud.google.com/edge-tpu

[30] "Jetson Modules, Support, Ecosystem, and Lineup | NVIDIA Developer." Accessed: Nov. 03, 2023. [Online]. Available: https://developer.nvidia.com/embedded/jetson-modules

[31] "What Is Apple's Neural Engine and How Does It Work?" Accessed: Nov. 03, 2023. [Online]. Available: https://www.makeuseof.com/what-is-a-neural-engine-how-does-it-work/

[32] S. Karaman et al., "Project-based, collaborative, algorithmic robotics for high school students: Programming self-driving race cars at MIT," 2017 IEEE Integrated STEM Education Conference (ISEC), Princeton, NJ, USA, 2017, pp. 195-203, doi: 10.1109/ISECon.2017.7910242.

[33] "Sensor-rich autonomous vehicle - the QCar from Quanser." Accessed: Nov. 05, 2023. [Online]. Available: https://www.quanser.com/products/qcar/

[34] "Semantic Segmentation Datasets for Autonomous Driving | HackerNoon." https://hackernoon.com/semantic-segmentation-datasets-for-autonomousdriving-1182ebd2aff0 (accessed Oct. 28, 2023).

[35] "Cityscapes Dataset – Semantic Understanding of Urban Street Scenes." Accessed: Nov. 03, 2023. [Online]. Available: https://www.cityscapes-dataset.com/

[36] "Mapillary." Accessed: Nov. 03, 2023. [Online]. Available: https://www.mapillary.com/dataset/vistas

[37] "Papers with Code - DUS Dataset." Accessed: Nov. 03, 2023. [Online]. Available: https://paperswithcode.com/dataset/dus

[38] "Papers with Code - CamVid Dataset." Accessed: Nov. 03, 2023. [Online]. Available: https://paperswithcode.com/dataset/camvid

[39] "The KITTI Vision Benchmark Suite." Accessed: Nov. 03, 2023. [Online]. Available: https://www.cvlibs.net/datasets/kitti/

[40] E. Xie, W. Wang, Z. Yu, A. Anandkumar, J. M. Alvarez, and P. Luo, "SegFormer: Simple and Efficient Design for Semantic Segmentation with Transformers." arXiv, Oct. 28, 2021. doi: 10.48550/arXiv.2105.15203.

[41] V. Badrinarayanan, A. Kendall, and R. Cipolla, "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation." arXiv, Oct. 10, 2016. doi: 10.48550/arXiv.1511.00561.