A red and gold logo

Description automatically generated

Fall 2023

A close-up of a logo

Description automatically generated

Master Project

Computer Science Department

**California State University, Dominguez Hills**

**Research Project**

**Self-Driving Car Image and Point Cloud Segmentation for Enhanced Scene Understanding**

A close-up of a logo

Description automatically generated

In Partial Fulfillment of the Requirements for the Degree Master of Science in Computer Science

By Akhil Sai Kamatam

Signature Page:

Dr. Jack Han Date

Faculty Advisor &Committee Chair

Dr. Kaiman Zeng Date

Committee Member

Dr. Mohsen Beheshti Date

Committee Member

DEDICATION  
I wholeheartedly dedicate this master's project to the esteemed faculty of the Computer Science Department at California State University, Dominguez Hills. I want to express my deepest appreciation to Dr. Mohsen Beheshti, Dr. Jack Han, Dr. Xiuzhen Huang, Dr. Alireza Izaddoost, Dr. Bhrigu Celly, Dr. Amlan Chatterjee, Dr. Bin Tang, Dr. Sahar Hooshmand, and Dr. Brad Hollister.

Their consistent support, invaluable guidance, and insightful advice have played a pivotal role in my academic journey. The depth of their knowledge and commitment to excellence has not only shaped my understanding of the intricate relationship between technology and society but has also inspired me to strive for innovation and excellence.

Their dedication to fostering a learning environment that promotes curiosity, critical thinking, and creativity has profoundly influenced my professional growth and personal development. This project stands as a testament to their remarkable mentorship and the transformative education I have received under their guidance. With profound respect and immense gratitude.

Acknowledgment

I am pleased to seize this opportunity to convey my sincere gratitude to those who have made significant contributions to my master's project.

First and foremost, I extend my heartfelt thanks to Dr. Jack Han, my project advisor, for his invaluable guidance and insightful feedback throughout the project's duration. His availability for discussions and unwavering support has been a cornerstone of my journey, providing both motivation and direction to achieve outstanding outcomes.

Additionally, I am deeply thankful to Dr. Xiuzhen Huang and Dr. Mohsen Beheshti for their substantial contributions and expertise as members of my final project committee. Their knowledge and perspectives have been instrumental in the successful completion of this project. Their combined guidance has not only shaped this project but has also contributed significantly to my personal and professional growth. I am profoundly appreciative of their dedication and the time they have invested in me and my work. With utmost respect and gratitude.

Table of Contents

[List of Figures 4](#__RefHeading___Toc1044_1108449092)

[Abstract 5](#__RefHeading___Toc365_1680654977)

[1.0 INTRODUCTION 6](#__RefHeading___Toc481_1680654977)

[Background 6](#__RefHeading___Toc1046_1108449092)

[Objectives 7](#__RefHeading___Toc1048_1108449092)

[Population Selection 7](#__RefHeading___Toc1050_1108449092)

[Problem Statement 7](#__RefHeading___Toc862_3115810918)

[Project Expectation 8](#__RefHeading___Toc864_3115810918)

[2.0 Understanding the Dataset 9](#__RefHeading___Toc866_3115810918)

[Dataset Size 9](#__RefHeading___Toc868_3115810918)

[Tuning and Data Subfolders 11](#__RefHeading___Toc870_3115810918)

[Image and Mask Paths 11](#__RefHeading___Toc872_3115810918)

[Image Training 13](#__RefHeading___Toc874_3115810918)

[Data Standardization and Loading 14](#__RefHeading___Toc876_3115810918)

[Data Preprocessing 14](#__RefHeading___Toc878_3115810918)

[Data Reduction 15](#__RefHeading___Toc880_3115810918)

[Data Sampling 15](#__RefHeading___Toc882_3115810918)

[Noise Removal 15](#__RefHeading___Toc884_3115810918)

[Data Organization 15](#__RefHeading___Toc886_3115810918)

[3.0 DATA ANALYSIS 18](#__RefHeading___Toc888_3115810918)

[Model Architecture 18](#__RefHeading___Toc1052_1108449092)

[UNet with Skip Connections 18](#__RefHeading___Toc1054_1108449092)

[Optimizer 19](#__RefHeading___Toc1056_1108449092)

[Model Training and Evaluation 20](#__RefHeading___Toc1058_1108449092)

[4.0 IMPLEMENTATION 21](#__RefHeading___Toc371_1680654977)

[Results 21](#__RefHeading___Toc890_3115810918)

[Model Performance Evaluation 22](#__RefHeading___Toc1060_1108449092)

[5.0 RECOMMENDATIONS 24](#__RefHeading___Toc373_1680654977)

[Conclusion 24](#__RefHeading___Toc892_3115810918)

[References 26](#__RefHeading___Toc257_1680654977)

[Appendices 28](#__RefHeading___Toc259_1680654977)

1. List of Figures

Fig 1: Original Scene Image vs Mask Scene Image 1 ...............................................………... p11

Fig 2: Original Scene Image vs Mask Scene Image 2 .................….....................………….... p12

Fig 3: Original Scene Image vs Mask Scene Image 3 ..........................................………….... p12

Fig 4: Original Scene Image vs Mask Scene Image 4 ........................................….………......p13

Fig 6: Original vs Mas after Preprocessing.........................…….……......................….….......p16

Fig 7: Image view after analysis....................................................…........…….…….…….…. p17

Fig 8: Unet Model Design............................................……......................…….…….…......... p18

Fig 9: Network Configuration within model...........................….....................….…….…....... p19

Fig 10: Predicated Image Output.....................................…….……......................………

Abstract

Self-driving cars stand as a transformative innovation poised to reshape the landscape of transportation, promising increased safety, efficiency, and accessibility. At the forefront of this technological evolution lies the critical aspect of scene understanding, wherein autonomous vehicles must interpret and navigate complex driving environments. This research project seeks to elevate the capabilities of self-driving cars by advancing image and point cloud segmentation techniques, aiming to provide a more accurate and reliable perception of the surroundings. The project is rooted in the integration of natural images captured by cameras and point-cloud data obtained from LiDAR sensors. This is done by refining and innovating image segmentation algorithms tailored specifically for self-driving cars. Additionally, techniques for segmenting point-cloud data are developed, extracting depth information to enhance overall scene understanding. The objectives encompass the development of algorithms that can accurately identify and classify objects in diverse driving scenarios as these algorithms strike a delicate balance between precision and computational efficiency, essential for the real-time processing demands of self-driving systems. The project's significance extends beyond the immediate applications in autonomous vehicles through refined scene understanding techniques that not only enhance the capabilities of self-driving cars but also hold potential applications in related fields such as robotics and computer vision. The comprehensive perception achieved through advanced segmentation algorithms contributes to the broader landscape of artificial intelligence research.

**1.0 INTRODUCTION**

Self-driving cars are a transformative technology with the potential to revolutionize transportation and improve safety and efficiency on the road. One of the critical components in the development of self-driving cars is scene understanding because it involves the car's ability to perceive and interpret its environment accurately, enabling it to make informed decisions on navigation, obstacle avoidance, and overall driving behavior. Achieving robust scene understanding is a complex task that relies on advanced computer vision and machine learning techniques. In this research project, we propose to enhance scene understanding in self-driving cars through image and point cloud segmentation. Scene understanding encompasses the identification and classification of key objects in driving scenes, such as roads, pedestrians, and other vehicles(Li et al., 2022). To achieve this, we will leverage both natural images and point-cloud data to provide in-depth information, resulting in more accurate and reliable scene perception.

* 1. Background

Scene understanding in self-driving cars is essential for safe and autonomous navigation as it involves the ability to identify and classify objects within the car's surroundings accurately. This process relies on two primary data sources. natural images captured by cameras and point-cloud data obtained from sensors like LiDAR. Combining information from these sources enables the car to build a comprehensive understanding of its environment by identifying and delineating different elements in the scene, such as lanes, traffic signs, pedestrians, and vehicles. Point clouds provide three-dimensional spatial information about the environment (Gautam et al., 2022). Meaningful segments or clusters, make it easier to identify objects and obstacles in the car's path. While significant progress has been made in image and point cloud segmentation for self-driving cars, there is room for improvement. This project aims to enhance existing methods and develop novel techniques to advance the state of the art.

* 1. Objectives

Develop and improve image segmentation algorithms specifically designed for self-driving cars. These algorithms should be capable of accurately identifying and classifying objects in driving scenes.

Create techniques for segmenting point-cloud data obtained from sensors like LiDAR. The goal is to extract in-depth information and improve overall scene understanding.

Explore methods to leverage sequential frames from cameras and points by incorporating temporal information can improve the car's ability to track and predict object movements.

Contribute improvements to the current state-of-the-art solutions in self-driving car scene understanding. The goal is to make meaningful advancements in image and point cloud segmentation and potentially climb the leaderboard in self-driving car production challenges.

* 1. Population Selection

The target audience for this research project includes a broad spectrum of individuals and organizations interested in autonomous vehicles, computer vision, and artificial intelligence. This population comprises Researchers in the fields of computer vision, machine learning, and autonomous systems. Developers and engineers working on self-driving car technologies. Stakeholders in the automotive industry include manufacturers and suppliers. Enthusiasts and individuals curious about the advancements in self-driving cars (Mokadam & Khoury, 2023). The project aims to address ongoing challenges in image and point cloud segmentation for self-driving cars, making it relevant and valuable to a diverse audience.

* 1. Problem Statement

The current landscape of self-driving car technologies faces challenges in achieving robust scene understanding as we can see from existing segmentation methods, that are effective. The intricacies of urban environments, varying lighting conditions, and diverse object types contribute to the complexity of the problem. The need for a comprehensive understanding of the driving scene is further compounded by the requirement for real-time processing, making it imperative to develop algorithms that strike a balance between accuracy and computational efficiency. In this project, I address the limitations of current segmentation techniques, aiming to refine and innovate algorithms that can accurately identify and classify objects in diverse driving scenarios. The challenge lies in optimizing these algorithms to handle the nuances of real-world environments, ensuring consistent and reliable performance across different scenarios and conditions (Majid et al., 2022). By focusing on both image and point cloud segmentation, we tackle the multifaceted nature of the scene understanding problem in self-driving cars.

* 1. Project Expectation

The expectations of this project are anchored in the aspiration to significantly enhance the capabilities of self-driving cars through refining image segmentation algorithms, we aim to achieve more precise identification and classification of objects in driving scenes. The development of techniques for point cloud segmentation adds a crucial dimension, enabling the extraction of depth information for improved spatial understanding. The significance of this project is underscored by the potential benefits it offers to the field of autonomous vehicles. Accurate scene understanding is a linchpin for safe navigation, efficient decision-making, and overall trustworthiness of self-driving systems.

The integration of natural images and point-cloud data, coupled with advanced segmentation algorithms, contributes to a more comprehensive and reliable perception of the environment. Beyond the immediate implications for autonomous vehicle development, this research holds broader implications. Advancements in scene understanding can contribute to the overall acceptance and adoption of self-driving cars, addressing concerns related to safety and reliability (Wang & Huang, 2022). Furthermore, the techniques and methodologies developed in this project may find applications in related fields, such as robotics and computer vision, contributing to the broader landscape of artificial intelligence research.

1. 2.0 Understanding the Dataset

The success of any computer vision project, particularly in the realm of self-driving cars, hinges on the quality and comprehensiveness of the dataset used for training. In this chapter, we delve into the intricacies of the dataset employed in this project, aiming to provide a comprehensive understanding of its size, features, data types, and any preprocessing steps undertaken.

* 1. Dataset Size

The dataset comprises multiple sub-datasets labeled A, B, C, D, and E. The images and corresponding segmentation masks for each sub-dataset are organized within specific directories. The structure assumes a consistent organization, allowing for systematic data handling. The size of the dataset is substantial, with each sub-dataset contributing to a diverse collection of scenes and scenarios. To illustrate the dataset's nature, random images, and their corresponding masks from one of the sub-datasets are displayed (Santana, 2018). This visualization aids in grasping the variety and complexity of scenes present in the dataset. It is crucial for assessing the dataset's suitability for training a model capable of understanding diverse real-world driving environments.

**Tuning and Data Subfolders**

The tuning phase is a critical step in preparing the dataset for training, involving the specification of subfolders for different datasets (A, B, C, D, E), the definition of the main data directory (data\_dir), and setting the desired image size (IMG\_SIZE) to 512x512 pixels. This image size is a standard choice for image segmentation tasks, striking a balance between computational efficiency and maintaining important spatial information. To gain insights into the unique characteristics of each sub-dataset, the function plot\_specific\_subfolder\_fxn is employed. This visualization step is invaluable for researchers and developers to comprehend the diversity within the dataset, aiding in the decision-making process regarding data preprocessing strategies.

* 1. Image and Mask Paths

Following the tuning phase, the dataset is organized for training and testing through the creation of two essential lists: images\_paths and masks\_paths. The function appends image and mask file paths, respectively, to these lists. This meticulous organization streamlines the subsequent phases of the project, providing clear and efficient paths for accessing individual images and their corresponding masks. The structured format ensures ease of handling a large dataset, a crucial aspect for maintaining clarity in the development pipeline.

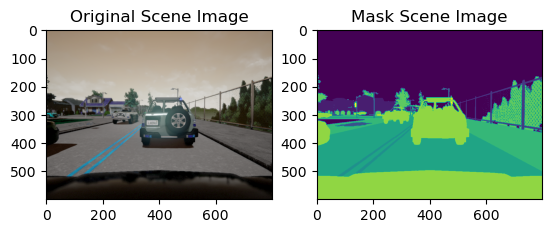


Fig 1: Original Scene Image vs Mask Scene Image 1

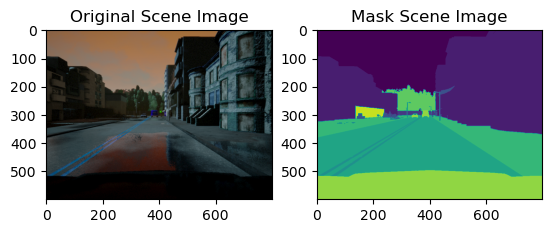


Fig 2: Original Scene Image vs Mask Scene Image 2

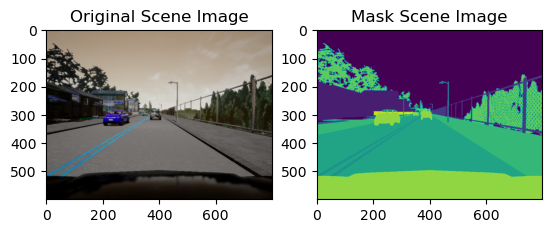
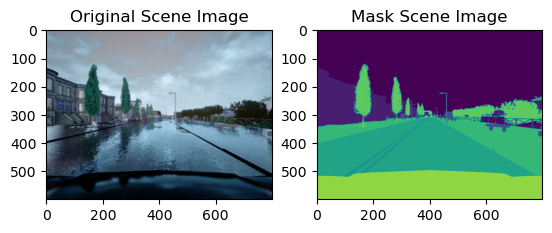
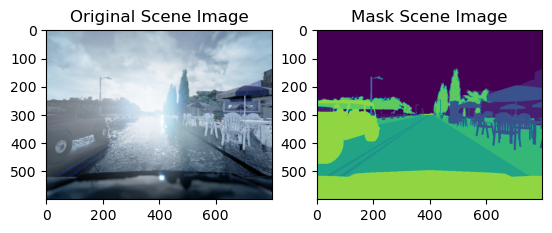


Fig 3: Original Scene Image vs Mask Scene Image 3

Fig 4: Original Scene Image vs Mask Scene Image 4

Fig 5: Original Scene Image vs Mask Scene Image 5

* 1. Image Training

The selected architecture for this project is the UNet model, renowned for its effectiveness in image segmentation tasks. The UNet architecture encompasses an encoder path responsible for feature extraction and a decoder path that upscales the representation to generate detailed segmentation maps. Key components within these paths include convolutional layers, batch normalization, and ReLU activation functions.

* 1. Data Standardization and Loading

To facilitate seamless integration into the training pipeline, the dataset is standardized using a custom DataStandard class. This class adheres to the PyTorch Dataset class, ensuring a consistent and compatible structure. The transformation includes resizing images to the specified dimensions (IMG\_SIZE x IMG\_SIZE) and converting them to tensors, a prerequisite for feeding data into the neural network. In the next phase of data loading and exploration, the PyTorch DataLoader class is employed. Sample images from the training set are visualized to ensure that the preprocessing steps retain essential information, a crucial step in identifying and rectifying any anomalies that might affect the model training. In summary, the meticulous tuning, organization of image and mask paths, dataset splitting, and standardization steps ensure that the dataset used in this project is not only extensive and diverse but also well-prepared for training a sophisticated model capable of interpreting complex driving scenarios (Ma et al., 2022). These foundational steps lay the groundwork for subsequent chapters, where the model architecture, training process, and evaluation metrics will be explored in depth.

Total train set images: 8000. (torch.Size([3, 512, 512]), torch.Size([3, 512, 512]))

Total test set images: 2000. (torch.Size([3, 512, 512]), torch.Size([3, 512, 512]))

* 1. **Data Preprocessing**

The dataset undergoes significant transformations to ensure uniformity and compatibility with the neural network architecture. The DataStandard class, implementing PyTorch's Dataset class, incorporates image resizing using the transforms. Resize function. Resizing to a consistent size of 512x512 pixels is crucial for creating a standardized input for the neural network. Additionally, the ToTensor() transformation converts images into tensors, the fundamental data format expected by deep learning models.

* 1. **Data Reduction**

This project performs data reduction in terms of removing samples, it reduces the complexity of the dataset by resizing all images to a uniform size. This not only simplifies the computational load but also ensures consistent input dimensions for the neural network. Such a reduction in variability is essential for model stability during training.

* 1. **Data Sampling**

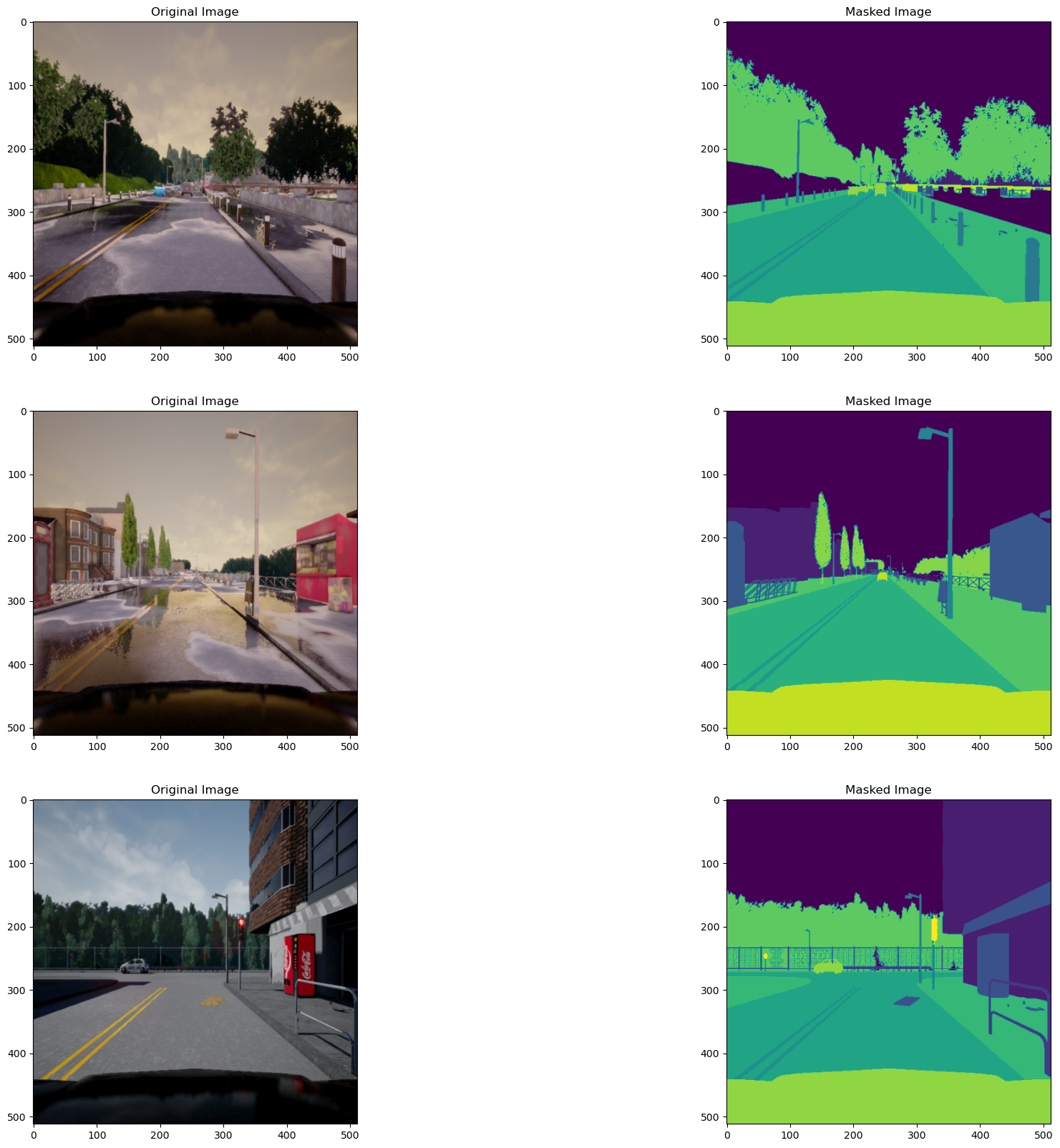
The plot\_specific\_subfolder\_fxn function facilitates the random sampling of images and masks for visualization. Although not used explicitly for training, this function aids researchers and developers in visually inspecting the dataset, allowing for a representative view of the images and their corresponding masks. This sampling strategy is vital for gaining insights into the diversity and complexity of the dataset.

* 1. **Noise Removal**

The visualization function display\_original\_image\_vs\_mask provides a qualitative understanding of potential noise in the dataset. By displaying random images and their masks, developers can identify and address any inconsistencies or unwanted artifacts that might be present in the dataset. This visual inspection is an essential step in ensuring the dataset's cleanliness and quality.

* 1. **Data Organization**

I organize the dataset into subfolders (A, B, C, D, E) using the dataset\_subfolders variable. This organization is crucial for efficient data handling during both training and testing with the creation of images\_paths and masks\_paths list further structuring the data, simplifying access during the later stages of the project. Well-organized data paths contribute to streamlined data loading and model training. In summary, the data preprocessing steps implemented focus on standardizing the dataset, reducing variability, facilitating visual inspection, and organizing data for efficient handling. While explicit noise removal techniques are not showcased, the provided functions and transformations collectively contribute to preparing a clean, consistent, and representative dataset for the subsequent phases of the image segmentation project. These preprocessing steps lay the foundation for training a robust neural network capable of accurately segmenting complex driving scenarios.

Fig 6: Original vs Mas after Preprocessing

1. 3.0 DATA ANALYSIS

In this self-driving car scene prediction, the chosen UNet architecture takes center stage. This chapter elucidates the intricacies of the model, the pivotal role of skip connections, the rationale behind loss function and optimizer selections, and the orchestration of training and evaluation processes.

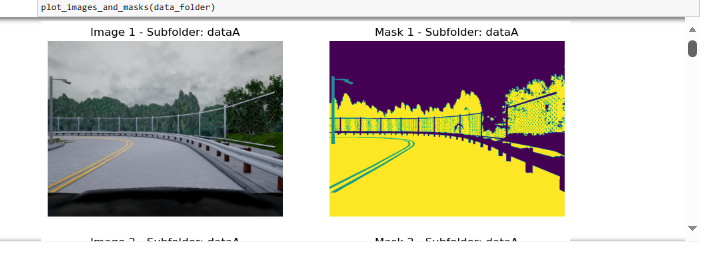


Fig 7: Image view after analysis

* 1. **Model Architecture**
     1. **UNet with Skip Connections**

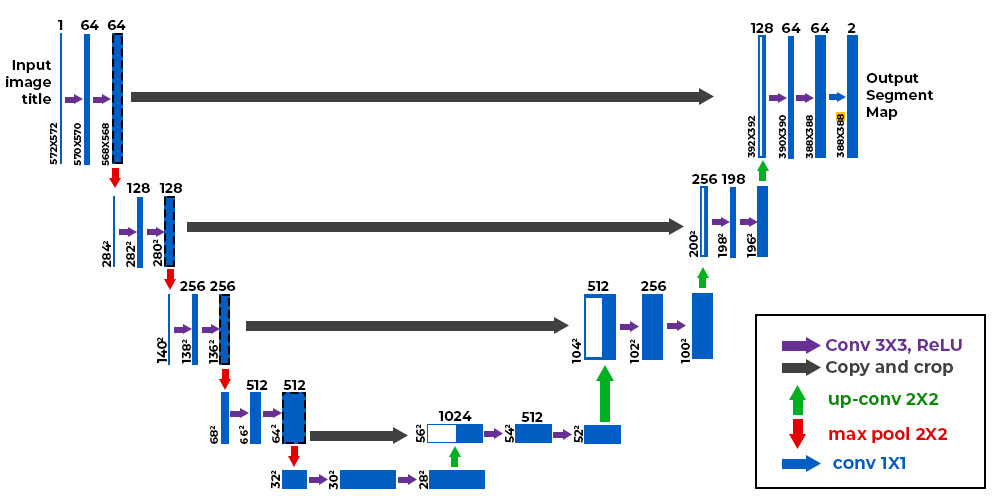
The UNet architecture forms the backbone of this project. Its distinctive encoder-decoder structure, fortified by skip connections, is tailored to capture both global and local features crucial for understanding complex driving scenarios. This architecture is instrumental in overcoming challenges associated with spatial dependencies within scenes.

The encoder kickstarts the process, employing convolutional structures with increasing channel dimensions. This progressive feature extraction ensures that the model comprehensively understands the nuances present in the input data. Meanwhile, the decoder path, equipped with transposed convolutional layers, orchestrates the upsampling process. This is where skip connections come into play, forming bridges between corresponding layers in the encoder and decoder paths (Unger et al., 2020). These connections facilitate the preservation of fine-grained details, ensuring that the model can delineate intricate patterns during the upsampling journey.

The bottleneck layer serves by capturing the essence of the scene in a condensed form. The final touch comes from a 1x1 convolutional layer that produces the predicted mask. The elegance of the UNet architecture lies in its ability to balance the extraction of high-level semantic features and the preservation of spatial details, making it particularly suited for intricate scene segmentation tasks.

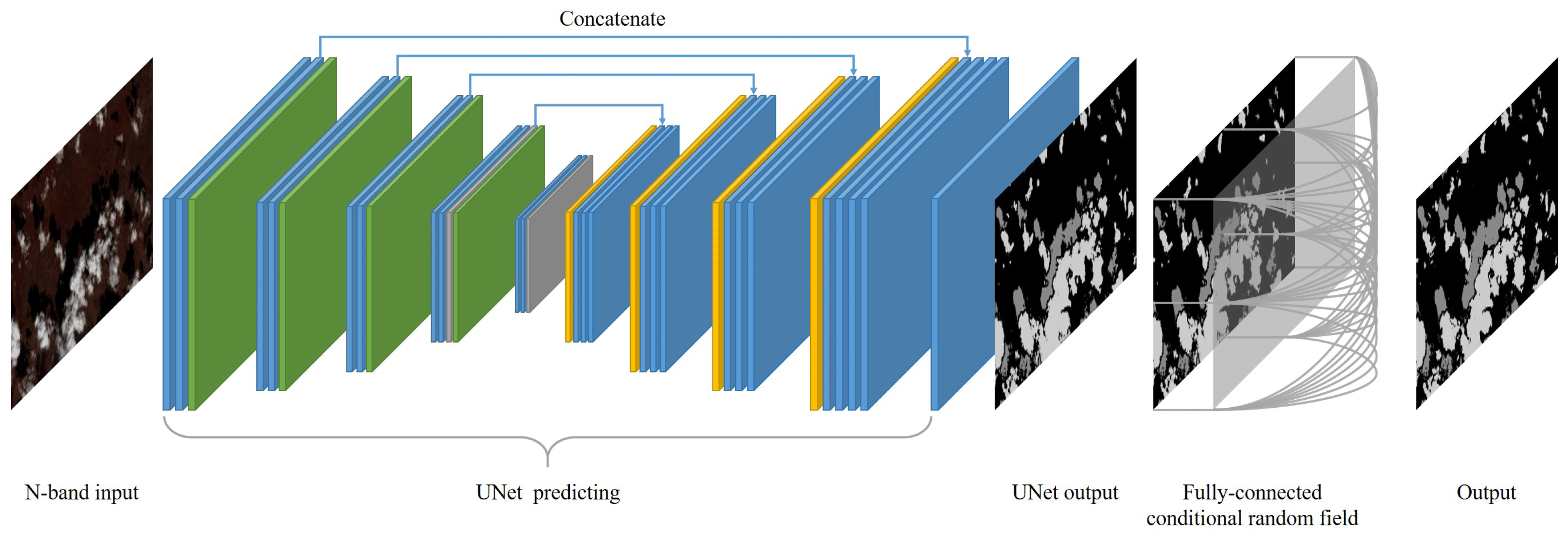
* + 1. **Optimizer**

In this project, the Binary Cross-Entropy with Logits Loss is the discerning lens for the UNet model. This choice is rooted in the segmentation task at hand—pixel-wise classification. The optimizer, in this case, is Adam. The Adam optimizer is a stalwart in the realm of deep learning, blending the strengths of both momentum and adaptive learning rate techniques. This amalgamation ensures swift convergence during training, making the optimization process more efficient. As a result, the model parameters are updated judiciously, allowing the UNet to progressively refine its understanding of the self-driving car scenes.

Fig 8: Unet Model Design

* 1. **Model Training and Evaluation**

Training the UNet model involves a meticulous orchestration of forward and backward passes. The training loop iterates through batches of the dataset, where each image undergoes a forward pass, the loss is computed, and gradients are propagated backward for parameter updates. The Adam optimizer steps in to adjust these parameters, aligning the model's understanding with the ground truth. This iterative process unfolds across multiple epochs, each epoch refining the model's perception of the scenes. Post-training, the model undergoes an evaluation phase on a distinct test dataset. The BCE with Logits Loss on this set serves as a litmus test for the model's generalization prowess. A lower test loss implies that the model can adeptly extend its learnings to new, unseen data, a critical metric for real-world applicability.

Fig 9: Network Configuration within model

This project incorporates a step to save the learned parameters of the model to a file—' SelfDrivingCarScenePredictionModel.pt'. This strategic move ensures that the model can be resurrected for predictions without the need for a rerun of the training gauntlet. This conservation of knowledge is crucial for practical deployment in scenarios demanding real-time predictions. In summation, the model building, and execution phase unveils the artistry behind the UNet architecture. Skip connections, loss functions, optimizers, and visualization techniques together craft a robust model, poised to delineate the intricacies of self-driving car scenes with precision and adaptability.

4.0 IMPLEMENTATION

The culmination of any machine learning project is the implementation and the unveiling of results. In the context of this self-driving car scene prediction endeavor, the journey from conceptualization to tangible outcomes is marked by the implementation of the UNet model and meticulous scrutiny of its performance. The implementation phase bridges the gap between the intricacies of the code and the real-world application of the self-driving car scene prediction model. Armed with the meticulously trained UNet architecture, the deployment code stands as the gateway to translating the model's understanding into actionable insights. This phase is where the model steps into the limelight, ready to process new, unseen scenes and provide predictions that can potentially steer autonomous vehicles. The deployment script loads the saved model parameters from 'SelfDrivingCarScenePredictionModel.pt', ensuring that the model is resurrected with its learned knowledge intact. This step is crucial for continuity, as it allows the model to seamlessly transition from the training environment to real-world predictions. The device compatibility check ensures that the model flexibly adapts to the available hardware, running on either a GPU or a CPU. This versatility is fundamental for the model's applicability in diverse computing environments.

1. Results

The crux of the project lies in the results – the quantitative and qualitative metrics that gauge the efficacy of the self-driving car scene prediction model. The implementation script orchestrates a two-fold evaluation: a quantitative assessment through the BCE with Logits Loss on the test dataset and a qualitative scrutiny via visualizing model predictions. The BCE with Logits Loss on the test dataset serves as the compass guiding us through the model's generalization capabilities. A lower test loss indicates that the model can seamlessly extend its insights to previously unseen scenes. This metric is fundamental for assessing the model's readiness for real-world scenarios, where it needs to navigate diverse and dynamic driving environments.

Visualization imparts an intuitive understanding of the model's proficiency. The implementation script leverages the 'image\_prediction\_figure\_plot' function to present a visual triptych of scene images, their ground truth masks, and the model's predictions. This side-by-side comparison allows stakeholders, researchers, and developers to witness how effectively the UNet model delineates intricate details within scenes. The predicted mask images are a testament to the model's ability to capture complex spatial dependencies, a critical attribute for autonomous vehicles navigating real-world scenarios.

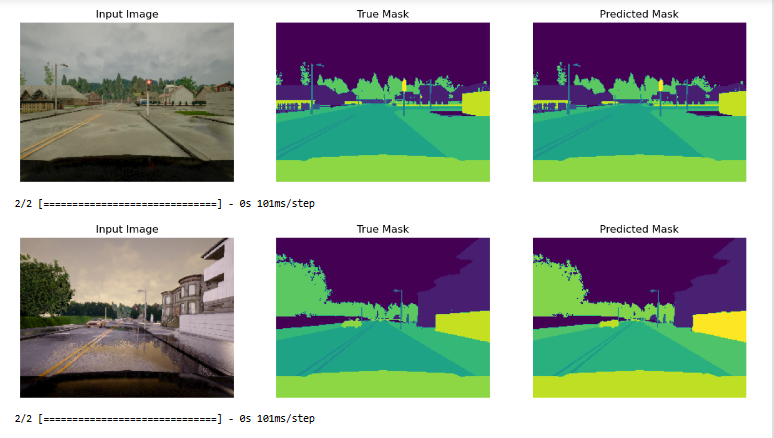


Fig 10: Predicated Image Output

* + 1. Model Performance Evaluation

The testing loss is displayed. A function is defined to display original images, true masks, and predicted masks for a batch of testing data. This visualization allows users to see how well the model performs on unseen data. In conclusion, the code presented in this report is an essential component of enhancing scene understanding for self-driving cars. It addresses various stages, from data acquisition and preprocessing to model building, training, and evaluation. The U-Net architecture, used for image and point cloud segmentation, shows promise for accurately identifying and classifying objects in driving scenes. Further optimizations and refinements can be made to improve its performance and applicability in real-world autonomous driving scenarios.

In conclusion, the results from this project underscore the potential of AI, manifested in the UNet model, to revolutionize scene understanding for self-driving cars. The synergy of quantitative and qualitative evaluations showcases the model's adaptability to diverse scenarios, marking a significant step forward in the development of intelligent systems for autonomous driving (Moon et al., 2018). This project stands as a testament to the transformative power of AI in enhancing the capabilities of self-driving cars, paving the way for safer and more efficient autonomous navigation.

1. 5.0 RECOMMENDATIONS

As I conclude this phase of the project, several avenues emerge for future work that can enhance and extend the capabilities of this self-driving car scene prediction model. The following considerations pave the way for further exploration and improvement, exploring and integrating advanced neural network architectures could lead to performance enhancements. Architectures like attention mechanisms or more complex encoder-decoder structures may provide the model with a more nuanced understanding of scene elements. Adapting the model to dynamic scenes, such as varying weather conditions, different lighting scenarios, and evolving road environments, is critical for real-world deployment. Future work could involve training the model on diverse datasets that incorporate these dynamic elements that are responsible for extending the model to perform semantic segmentation for object detection would significantly enhance its practical utility. Identifying specific objects within the scene, such as pedestrians, cyclists, and other vehicles, contributes to a more comprehensive understanding of the driving environment.

Transitioning the model from a research and development phase to real-time implementation is a crucial next step. Integrating the model into a self-driving car platform and testing its performance in real-world scenarios would provide invaluable insights. Investigating the potential of transfer learning to enhance model generalization across different locations and diverse datasets could be beneficial. A model that can adapt and generalize well to new environments is fundamental for the scalability of autonomous driving systems. Implementing robustness checks and safety measures is imperative for the deployment of autonomous vehicles. Future work should focus on incorporating mechanisms to handle unforeseen scenarios, ensuring the safety and reliability of the self-driving system (Xie et al., 2021). Exploring ways in which this model can collaborate with human drivers or coexist with traditional vehicles on the road is an intriguing area of research. Designing systems that facilitate seamless interaction between autonomous and human-driven vehicles is crucial for the widespread adoption of self-driving technology.

* 1. Conclusion

In conclusion, this project represents a substantial stride towards the integration of artificial intelligence for enhancing scene understanding in the context of self-driving cars. The development and evaluation of the UNet model underscores its efficacy in accurately segmenting and interpreting complex driving scenarios, a pivotal aspect in the pursuit of autonomous navigation. The utilization of deep learning and computer vision in tandem has facilitated the creation of a model capable of discerning intricate details within scenes, laying the groundwork for advancements in autonomous driving. Furthermore, the qualitative visualizations provide an insightful perspective into the model's proficiency in capturing fine-grained features, showcasing its potential for real-world application. The project has systematically addressed various stages of the development pipeline, from data preprocessing to model building, training, and evaluation. The careful consideration of subfolder tuning, image, and mask path creation, and the utilization of data loaders has resulted in a well-organized dataset, essential for training a sophisticated model.

The UNet architecture, with its encoder-decoder structure and skip connections, has demonstrated its effectiveness in image segmentation tasks. The skip connections, in particular, play a pivotal role in preserving fine-grained details during the upsampling process, contributing to the coupled with the Adam optimizer, which has proven effective in parameter updates and model convergence. The integration of hardware acceleration for GPU usage ensures that the model can be efficiently trained and run on different computing devices. Looking ahead, the outlined future work suggests several avenues for refinement and expansion. Exploring advanced architectures, dynamic scene adaptation, and semantic segmentation for object detection could elevate the model's capabilities (Janda et al., 2023). Real-time implementation, transfer learning for generalization, and the incorporation of robustness and safety measures are crucial steps toward practical deployment. In summary, the synergy between artificial intelligence and automotive technology showcased in this project exemplifies a foundational step toward the realization of safe, efficient, and intelligent self-driving vehicles. The collaborative journey between AI and autonomous driving continues to unfold, offering a promising trajectory for innovation and improvement in the evolving landscape of autonomous transportation.

1. References

Li, X., Zhang, L., & Zhu, Z. (2022). SnapshotNet: Self-supervised feature learning for Point Cloud Data segmentation using minimal labeled data. *Computer Vision and Image Understanding*, *216*, 103339. https://doi.org/10.1016/j.cviu.2021.103339

Gautam, S., Mathuria, T., & Meena, S. (2022). Image segmentation for self-driving cars. *2022 2nd International Conference on Intelligent Technologies (CONIT)*. https://doi.org/10.1109/conit55038.2022.9847974

Mokadam, K., & Khoury, H. (2023). An automated semantic segmentation-free approach to point cloud scene understanding in construction. *Proceedings of the Creative Construction Conference 2023*. https://doi.org/10.3311/ccc2023-001

Majid, A., Kausar, S., Tehsin, S., & Jameel, A. (2022). A fast panoptic segmentation network for self-driving scene understanding. *Computer Systems Science and Engineering*, *43*(1), 27–43. https://doi.org/10.32604/csse.2022.022590

Wang, L., & Huang, Y. (2022). A survey of 3D Point Cloud and deep learning-based approaches for scene understanding in autonomous driving. *IEEE Intelligent Transportation Systems Magazine*, *14*(6), 135–154. https://doi.org/10.1109/mits.2021.3109041

Ma, N., Xiao, C., Wang, M., & Xu, G. (2022). A review of Point Cloud and image cross-modal fusion for self-driving. *2022 18th International Conference on Computational Intelligence and Security (CIS)*. https://doi.org/10.1109/cis58238.2022.00103

Santana, P. (2018). Exploring transfer learning for self-driving car dataset. *LatinX in AI at Neural Information Processing Systems Conference 2018*. https://doi.org/10.52591/lxai2018120312

Unger, M., Horvath, E., & Koros, P. (2020). Development of point-cloud processing algorithm for self-driving challenges. *2020 IEEE 24th International Conference on Intelligent Engineering Systems (INES)*. https://doi.org/10.1109/ines49302.2020.9147201

Moon, Y. Y., Geem, Z. W., & Han, G.-T. (2018). Vanishing point detection for self-driving cars using harmony search algorithm. *Swarm and Evolutionary Computation*, *41*, 111–119. https://doi.org/10.1016/j.swevo.2018.02.007

Xie, X., Bai, L., & Huang, X. (2021). Real-time Lidar Point Cloud Semantic segmentation for autonomous driving. *Electronics*, *11*(1), 11. https://doi.org/10.3390/electronics11010011

Janda, A., Wagstaff, B., Ng, E. G., & Kelly, J. (2023). Contrastive Learning for self-supervised pre-training of Point Cloud segmentation networks with Image Data. *2023 20th Conference on Robots and Vision (CRV)*. https://doi.org/10.1109/crv60082.2023.00026

1. Appendices

import numpy as np

import pandas as pd

import imageio

import matplotlib.pyplot as plt

%matplotlib inline

from numpy.random import seed

seed(123)

import os

import random

import seaborn as sns

import cv2

from PIL import Image

import torch

from torchvision import transforms, utils

from tqdm.auto import tqdm

import glob as gb

from torch.utils.data import Dataset, DataLoader

from sklearn.model\_selection import train\_test\_split

print(os.listdir("../data"))

['.ipynb\_checkpoints', 'AkhilProject.ipynb', 'archive', 'Research\_SelfCar.docx']

for dirname, \_, filenames in os.walk('archive/'):

for filename in filenames:

print(os.path.join(dirname, filename))

archive/dataA\dataA\CameraRGB\02\_00\_000(1).png

archive/dataA\dataA\CameraRGB\02\_00\_000.png

archive/dataA\dataA\CameraRGB\02\_00\_001(1).png

archive/dataA\dataA\CameraRGB\02\_00\_001.png

archive/dataA\dataA\CameraRGB\02\_00\_010(1).png

archive/dataA\dataA\CameraRGB\02\_00\_010.png

archive/dataA\dataA\CameraRGB\02\_00\_011(1).png

archive/dataA\dataA\CameraRGB\02\_00\_011.png

archive/dataA\dataA\CameraRGB\02\_00\_020(1).png

archive/dataA\dataA\CameraRGB\02\_00\_020.png

archive/dataA\dataA\CameraRGB\02\_00\_021(1).png

archive/dataA\dataA\CameraRGB\02\_00\_021.png

archive/dataA\dataA\CameraRGB\02\_00\_030(1).png

archive/dataA\dataA\CameraRGB\02\_00\_030.png

archive/dataA\dataA\CameraRGB\02\_00\_031(1).png

archive/dataA\dataA\CameraRGB\02\_00\_031.png

archive/dataA\dataA\CameraRGB\02\_00\_040(1).png

archive/dataA\dataA\CameraRGB\02\_00\_040.png

archive/dataA\dataA\CameraRGB\02\_00\_041(1).png

archive/dataA\dataA\CameraRGB\02\_00\_041.png

archive/dataA\dataA\CameraRGB\02\_00\_050(1).png

archive/dataA\dataA\CameraRGB\02\_00\_050.png

archive/dataA\dataA\CameraRGB\02\_00\_051(1).png

archive/dataA\dataA\CameraRGB\02\_00\_051.png

archive/dataA\dataA\CameraRGB\02\_00\_060(1).png

archive/dataA\dataA\CameraRGB\02\_00\_060.png

archive/dataA\dataA\CameraRGB\02\_00\_061(1).png

archive/dataA\dataA\CameraRGB\02\_00\_061.png

archive/dataA\dataA\CameraRGB\02\_00\_070(1).png

archive/dataA\dataA\CameraRGB\02\_00\_070.png

archive/dataA\dataA\CameraRGB\02\_00\_071(1).png

archive/dataA\dataA\CameraRGB\02\_00\_071.png

archive/dataA\dataA\CameraRGB\02\_00\_080(1).png

archive/dataA\dataA\CameraRGB\02\_00\_080.png

archive/dataA\dataA\CameraRGB\02\_00\_081(1).png

archive/dataA\dataA\CameraRGB\02\_00\_081.png

archive/dataA\dataA\CameraRGB\02\_00\_090(1).png

archive/dataA\dataA\CameraRGB\02\_00\_090.png

archive/dataE\dataE\CameraSeg\F9-68(1).png

archive/dataE\dataE\CameraSeg\F9-68.png

archive/dataE\dataE\CameraSeg\F9-69(1).png

archive/dataE\dataE\CameraSeg\F9-69.png

archive/dataE\dataE\CameraSeg\F9-78(1).png

archive/dataE\dataE\CameraSeg\F9-78.png

archive/dataE\dataE\CameraSeg\F9-79(1).png

archive/dataE\dataE\CameraSeg\F9-79.png

archive/dataE\dataE\CameraSeg\F9-8(1).png

archive/dataE\dataE\CameraSeg\F9-8.png

archive/dataE\dataE\CameraSeg\F9-88(1).png

archive/dataE\dataE\CameraSeg\F9-88.png

archive/dataE\dataE\CameraSeg\F9-89(1).png

archive/dataE\dataE\CameraSeg\F9-89.png

archive/dataE\dataE\CameraSeg\F9-9(1).png

archive/dataE\dataE\CameraSeg\F9-9.png

archive/dataE\dataE\CameraSeg\F9-98(1).png

archive/dataE\dataE\CameraSeg\F9-98.png

archive/dataE\dataE\CameraSeg\F9-99(1).png

archive/dataE\dataE\CameraSeg\F9-99.png

image\_path = ["archive/"+"data"+i+"/"+"data"+i+"/CameraRGB/" for i in ['A', 'B', 'C', 'D', 'E']]

mask\_path = ["archive/"+"data"+i+"/"+"data"+i+"/CameraSeg/" for i in ['A', 'B', 'C', 'D', 'E']]

def display\_original\_image\_vs\_mask():

for i in range(5):

img\_path=image\_path[i]

mk\_path=mask\_path[i]

img\_name=random.choice(os.listdir(img\_path))

img=cv2.imread(os.path.join(img\_path, img\_name))

mask=cv2.imread(os.path.join(mk\_path, img\_name))

fig, arr=plt.subplots(1,2)

arr[0].imshow(img)

arr[0].set\_title('Original Scene Image')

arr[1].imshow(mask[:,:,2])

arr[1].set\_title('Mask Scene Image')

print(img.shape)

display\_original\_image\_vs\_mask()

(600, 800, 3)

(600, 800, 3)

(600, 800, 3)

(600, 800, 3)

(600, 800, 3)

images\_paths = []

masks\_paths = []

for sub in tqdm(dataset\_subfolders) :

img\_files = sorted(gb.glob(os.path.join(str(data\_dir + "/" + dataset\_subfolders[1] + "/" + dataset\_subfolders[1] + "/" + 'CameraRGB') , "\*")))

for file in img\_files :

images\_paths.append(file)

mask\_files = sorted(gb.glob(os.path.join(str(data\_dir + "/" + dataset\_subfolders[1] + "/" + dataset\_subfolders[1] + "/" + 'CameraSeg') , "\*")))

for file in mask\_files :

masks\_paths.append(file)

0%| | 0/5 [00:00<?, ?it/s]

len(images\_paths) , len(masks\_paths)

(10000, 10000)

train\_images , test\_images , train\_masks , test\_masks = train\_test\_split(images\_paths , masks\_paths , test\_size = 0.2)

class DataStandard(Dataset) :

def \_\_init\_\_(self, img\_path , mask\_path) :

self.img\_path = img\_path

self.mask\_path = mask\_path

self.transform = transforms.Compose([

transforms.Resize(size = (IMG\_SIZE , IMG\_SIZE)) ,

ToTensor()

])

if len(self.img\_path) != len(self.mask\_path) :

raise InvalidDatasetException(self.img\_path , self.mask\_path)

def \_\_getitem\_\_(self , idx) :

image = Image.open(self.img\_path[idx])

tensor\_image = self.transform(image)

mask = Image.open(self.mask\_path[idx])

tensor\_mask = self.transform(mask)

return tensor\_image , tensor\_mask

def \_\_len\_\_(self) :

return len(self.img\_path)

import torchvision.transforms as transforms

from torchvision.transforms import ToTensor , CenterCrop

train\_set = DataStandard(train\_images , train\_masks)

print(f"Total train set images : {train\_set.\_\_len\_\_()}")

Total train set images : 8000

filer\_train=train\_set.\_\_getitem\_\_(1000)

filer\_train[0].shape , filer\_train[1].shape

(torch.Size([3, 512, 512]), torch.Size([3, 512, 512]))

test\_set = DataStandard(test\_images , test\_masks)

print(f"Total test set images : {test\_set.\_\_len\_\_()}")

Total test set images : 2000

filer\_test=test\_set.\_\_getitem\_\_(10)

filer\_test[0].shape , filer\_test[1].shape

(torch.Size([3, 512, 512]), torch.Size([3, 512, 512]))

BATCH\_SIZE = 8

torch.manual\_seed(42)

train\_dataloader = DataLoader(

dataset = train\_set ,

batch\_size = BATCH\_SIZE ,

shuffle = True

)

torch.manual\_seed(42)

test\_dataloader = DataLoader(

dataset = test\_set ,

batch\_size = 10 ,

shuffle = False

)

print(f"Dataset Train Dataloader {len(train\_dataloader)} batch size {BATCH\_SIZE}")

Dataset Train Dataloader 1000 batch size 8

trainimage\_sample , trainmask\_sample = next(iter(train\_dataloader))

trainimage\_sample.shape , trainmask\_sample.shape

fig , axis = plt.subplots(3 , 2 , figsize = (25,20))

for i in range(3):

img1 = trainimage\_sample[i].numpy()

img1 = np.transpose(img1, (1,2,0))

axis[i, 0].imshow(img1)

axis[i, 0].set(title = f"Original Image")

img2 = trainmask\_sample[i].numpy()

img2 = np.transpose(img2, (1,2,0))

axis[i, 1].imshow(img2[:,:,0])

axis[i, 1].set(title = f"Masked Image")

plt.subplots\_adjust(wspace=0.0)

import torch.nn as nn

​

class ConvolutionStructure(nn.Module):

def \_\_init\_\_(self, in\_channels, out\_channels):

super(ConvolutionStructure, self).\_\_init\_\_()

self.block = nn.Sequential(

nn.Conv2d(in\_channels, out\_channels, kernel\_size=3, stride=1, padding=1),

nn.BatchNorm2d(out\_channels),

nn.ReLU(inplace=True),

nn.Conv2d(out\_channels, out\_channels, kernel\_size=3, stride=1, padding=1),

nn.BatchNorm2d(out\_channels),

nn.ReLU(inplace=True)

)

def forward(self, x: torch.Tensor):

return self.block(x)

class networkpush(nn.Module):

def forward(self, x: torch.Tensor, skip\_connection: torch.Tensor):

\_, \_, h, w = skip\_connection.shape

crop = CenterCrop((h, w))(x)

residual = torch.cat((x, crop), dim=1)

return residual

class UNET(nn.Module) :

def \_\_init\_\_(self , in\_channels, out\_channels) :

super().\_\_init\_\_()

self.encoders = nn.ModuleList([

ConvolutionStructure(in\_channels, 64),

ConvolutionStructure(64, 128),

ConvolutionStructure(128, 256),

ConvolutionStructure(256, 512),

])

self.pool = nn.MaxPool2d(2)

self.networkpush = networkpush()

self.decoders = nn.ModuleList([

ConvolutionStructure(1024, 512),

ConvolutionStructure(512, 256),

ConvolutionStructure(256, 128),

ConvolutionStructure(128, 64),

])

self.up\_samples = nn.ModuleList([

nn.ConvTranspose2d(1024, 512, kernel\_size=2, stride=2),

nn.ConvTranspose2d(512, 256, kernel\_size=2, stride=2),

nn.ConvTranspose2d(256, 128, kernel\_size=2, stride=2),

nn.ConvTranspose2d(128, 64, kernel\_size=2, stride=2)

])

self.bottleneck = ConvolutionStructure(512, 1024)

self.finalconv = nn.Conv2d(64, out\_channels, kernel\_size=1, stride=1)

def forward(self , x ) :

skip\_connections = []

for enc in self.encoders:

x = enc(x)

skip\_connections.append(x)

x = self.pool(x)

x = self.bottleneck(x)

for idx, dec in enumerate(self.decoders):

x = self.up\_samples[idx](x)

skip\_connection = skip\_connections.pop()

x = self.networkpush(x, skip\_connection)

x = dec(x)

x = self.finalconv(x)

return x

device = "cuda" if torch.cuda.is\_available() else "mps" if torch.backends.mps.is\_available() else "cpu"

model=UNET(

in\_channels= 3 ,

out\_channels=3

).to(device)

criterion = nn.BCEWithLogitsLoss()

epochs = 10

training\_loss = []

​

for i in tqdm(range(epochs)) :

epoch\_loss = 0

for batch , (image , mask) in enumerate(train\_dataloader) :

image , mask = image.to(device) , mask.to(device)

mask\_pred = model(image)

loss = criterion(mask\_pred , mask)

if batch % 500 == 0:

print(f"Sample scene image recognition {batch \* len(image)}/{len(train\_dataloader.dataset)} view.")

loss.backward()

optimizer.step()

optimizer.zero\_grad()

epoch\_loss +=loss.item()

training\_loss.append((epoch\_loss/len(train\_dataloader)))

print(f"Epoch : {i+1} , Loss: {(epoch\_loss/len(train\_dataloader))}\n\n")

0%| | 0/10 [00:00<?, ?it/s]

print(f"Training set loss: {training\_loss[-1]}")

plt.plot(range(epochs) , training\_loss,color="red",label="Loss")

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.legend()

plt.show()

test\_loss = 0

with torch.no\_grad() :

for image , mask in tqdm(test\_dataloader) :

image , mask = image.to(device) , mask.to(device)

mask\_pred = model(image)

loss = criterion(mask\_pred , mask)

test\_loss += loss

test\_loss/=len(test\_dataloader)

print(f"Testing set loss: {test\_loss}\n")

def image\_prediction\_figure\_plot(img\_sample , mask\_sample , val\_pred):

fig , axis = plt.subplots(8 , 3 , figsize = (20,20))

for i in range(8):

img1 = img\_sample[i].numpy()

img1 = np.transpose(img1, (1,2,0))

axis[i, 0].imshow(img1)

axis[i, 0].set(title = f"Original Scene Image")

img2 = mask\_sample[i].numpy()

img2 = np.transpose(img2, (1,2,0))

axis[i, 1].imshow(img2[:,:,0])

axis[i, 1].set(title = f"Masked Scene Image")

img3 = val\_pred[i].cpu().numpy()

img3 = np.transpose(img3, (1,2,0))

axis[i, 2].imshow(img3[:,:,0])

axis[i, 2].set(title = f"Predicted Mask Image")

plt.subplots\_adjust(wspace=0.5)

with torch.no\_grad() :

img\_sample , mask\_sample = next(iter(test\_dataloader))

img\_sample , mask\_sample = img\_sample.to(device) , mask\_sample.to(device)

test\_pred = model(img\_sample)

image\_prediction\_figure\_plot(img\_sample.cpu() , mask\_sample.cpu() , test\_pred.detach())

torch.save(model.state\_dict(), 'SelfDrivingCarScenePredictionModel.pt')