

Real Time Image Segmentation for Self Driving Cars

Submitted in partial fulfillment of the requirements for the
award of
Bachelor of Engineering degree in Computer Science and Engineering

By

MEDAM VINOD REDDY (Reg.No - 39110579)

ALLA PRANAV REDDY (Reg.No - 39110044)



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SCHOOL OF COMPUTING

SATHYABAMA

**INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)**

**Accredited with Grade "A" by NAAC | 12B Status by UGC | Approved by AICTE
JEPPIAAR NAGAR, RAJIV GANDHI SALAI,
CHENNAI - 600119**

APRIL - 2023



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)

Accredited with Grade "A" by NAAC | 12B Status by UGC | Approved by AICTE

www.sathyabama.ac.in

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of **Medam Vinod Reddy (Reg.No-39110579)** and **Alla Pranav Reddy (Reg-No39110044)** who carried out the Project Phase-2 "**Real Time Image Segmentation for Self Driving Cars**" under my supervision from January 2023 to April 2023.

Internal Guide

Dr. T. JUDGI, M.E., Ph. D

Head of the Department

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



Submitted for Viva voce Examination held on 20-04-2023

Internal Examiner

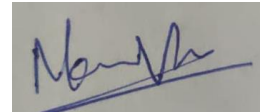
External Examiner

DECLARATION

I, **Medam Vinod Reddy (Reg.No- 39110579)**, hereby declare that the Project Phase-2 Report entitled “**Real Time Image Segmentation for Self Driving Cars**” done by me under the guidance of **Dr. T. JUDGI, M.E., Ph. D** is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering degree in **Computer Science and Engineering**.

DATE: 20-04-2023

PLACE: Chennai



SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

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

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

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

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

ABSTRACT

Predicting A few years ago, semantic segmentation was regarded as a difficult computer vision task. Because of recent advances in deep learning, relatively accurate solutions for its usage in automated driving are now conceivable. The majority of existing semantic segmentation algorithms are built for generic pictures and do not take into account prior structure or end objectives for automatic driving. The project describes an effective technique for semantic segmentation for self-driving cars. The development of an accurate and real-time semantic segmentation system for self-driving automobiles is urgently required. The initiative is based on self-driving automobiles, or autonomous driving. Such an application is required in autonomous driving, where self-driving vehicles must comprehend their surroundings, such as other automobiles, pedestrians, road lanes, traffic signs, or traffic lights. It is also known that the immense success of deep learning has had a significant influence on semantic segmentation approaches, enhancing their accuracy. A self-driving car must react rapidly to new occurrences in order to ensure the safety of passengers and other road users, however it is generally acceptable if object borders are not identified accurately down to a pixel resolution.

Table of Content

Chapter No	Title	Page No
	ABSTRACT	v
	LIST OF FIGURES	viii
1	INTRODUCTION	1
	1.1 Introduction	1
	1.2 History of semantic segmentation	1
	1.3 Pixel Level semantic segmentation	3
	1.4 Applications	3
2	LITERATURE SURVEY	6
	2.1 Inferences from the Literature Survey	12
	2.2 Open Problems in Existing System	12
3	REQUIREMENT ANALYSIS	13
	3.1 Feasibility Studies/Risk Analysis of the Project	13
	3.2 Software Requirements Specification Document	14
4	DESCRIPTION OF PROPOSED SYSTEM	15
	4.1 Selected Methodology or Process model	15
	4.2 Architecture/Overall Design of Proposed System	24
	4.3 Description of Software for Implementation and Testing Plan of the Proposed Model/System	24
	4.4 Project Management Plan	28
	4.5 Financial Report on Estimated Costing	28
	4.6 Transitions/Software to Operations Plan	29
5	IMPLEMENTATION DETAILS	31
	5.1 Development and Deployment Status	31
	5.2 Algorithms	32
	5.3 Testing	32

6	RESULT AND CONCLUSION	35
7	CONCLUSION	36
	7.1 Conclusion	36
	7.2 Future Work	37
	7.3 Research Issues	38
	7.4 Implementation Issues	38
	REFERENCES	39
	APPENDIX	41
	SCREENSHOTS	48

LIST OF FIGURES

Fig.no	Name	Page
1.1	Real time image Segmentation	4
4.1	Two stage Detector	16
4.2	Csp Bottleneck	18
4.3	Dense Net	19
4.4	Video input	23
4.5	System Architecture	24

CHAPTER-1

INTRODUCTION

1.1 Introduction

The standard of living is increasing rapidly with the development of computer technology. In most cases, people use computers to acquire information and resolve problems. An image, a visual presentation, is the easiest approach for humans to receive external information, which may contain diverse information. Being able to travel effectively and safely in driverless vehicles has been a hot topic of research over the past few years and numerous companies and research centers are trying to develop the first driverless car model that is completely convenient. In intelligent transport, self-driving requires not only that the system remains stable, but also that it efficiently recognizes the dynamic images of the road. This is a very promising area that offers many possible advantages, such as greater security, lower costs, comfortable travel, increased mobility and a reduced environmental footprint.

1.2 History of semantic segmentation

Over the past few years, several technology companies and universities have been very interested in autonomous vehicles or autonomous cars by investing immense resources both technical and financial for research and development. Some of them are going through a lot of research. They are Alphabet's subsidiary Waymo, Pilot Net of NVIDIA, Delphi with an MIT based start-up nuTonomy, General Motor's Cruise, Tesla 'S' model of Tesla Motors, BMW in collaboration with BAIDU, Ford's Agro AI, etc. Thus, the future of autonomous driving is imminent for its mainstream provided it overcomes technological and practical challenges with economic, social and legal acceptance. With rapid technological advances, computers are leaving behind the old notion of "computers are dumb machines". Of all the achievements obtained in various fields of information technology, an autonomous vehicle was one of the largest and most important inventions. This is a subject of research for many years and numerous algorithms involving advanced artificial intelligence concepts. The popularity of these cars is

on the rise. Recent research shows that about 10 million autonomous cars would be on the road in the near future. Companies like Audi, Volvo, Ford, Google, General Motors, BMW and Tesla have incorporated this technology in the latest versions.

Transportation accidents are one of the many causes of fatalities in the world. According to the report revealed by the “World Health Organization (WHO)”, over one million fatalities are caused because of road accidents and the numbers are even a lot of which incorporates very little or major injuries. Most accidents have to do with human error. Humans make mistakes in many ways, including using a mobile phone while driving, not following traffic rules, being distracted by billboards and not getting enough sleep, which usually results in drowsiness while driving. As a result, there are accidents as a result of these conditions. It is therefore necessary to come up with a solution that helps humans to drive safely. The history of autos began in the 1920s. Once the first car guidance was introduced, leading to further improvements and improvements in the cars. Additionally, the vision-guided vehicle was introduced in 1988 with the use of LIDAR and computer vision for tracking and detecting and preventing obstacles. The project has been funded in the United States through DARPA using emerging technologies. For around 20 years, “Uber”, “Tesla”, “google”, “Toyota” are some of the manufacturers that have been designing and testing these cars and they have achieved good results while moving towards complete automation.

Recently, the rapid development of AI has greatly enhanced the advancement of unmanned driving, such as autonomous cars, unmanned aerial vehicles, etc. Among these unmanned driving technologies, autonomous cars are getting more and more attention because of their significant economic impact. But there are a lot of challenges when it comes to autonomous cars. For example, the safety issue is the key technology that must be effectively addressed in self-driving cars; otherwise, self-driving cars cannot be allowed on the road. Deep learning is an important part of machine learning and has been a hot subject of late. Thanks to its excellent performance, it has been used by scientists to research and develop autonomous vehicles. More and more deep learning solutions for autonomous cars

have been presented, including barrier detection, scene recognition, lane detection, etc.

1.3 Pixel Level semantic segmentation

Semantic segmentation involves assigning each pixel of an image to a matching class label from a set of predefined categories. Although this may be considered as a pixel classification issue in an image, it is a much more complex procedure, compared to the standard classification aimed at predicting the label of the entire image. The enormous success of deep learning has had an enormous impact on semantic segmentation methods, enhancing their performance in terms of accuracy. These promising developments have generated interest in many fields of technology and research that require high-level computer vision capabilities. Such an application is autonomous driving, in which autonomous vehicles should understand their surroundings, i.e., other cars, pedestrians, traffic routes, traffic signs or traffic signals. Deep learning-based semantic segmentation is a key choice to achieve this objective, because of the phenomenal precision of deep neural networks in multi-class detection and recognition tasks.

1.4 Applications

Nevertheless, in applications, such as autonomous driving, where low latency operations are required, the cost of calculating these methods remains rather limited. Self-driving is one such application due to the critical need to make decisions at specific intervals. There is therefore a need to improve the design of segmentation models to achieve efficient architectures that will be able to operate in real time with the appropriate accuracy.

A number of image segmentation approaches exist. But authoritative segmentation standards have not yet been developed. Traditional segmentation algorithms depend on the attributes of the pixel itself. In effect, certain algorithms are combined with a specific field. However, these algorithms only deliver performance in specific areas or for specific image features.

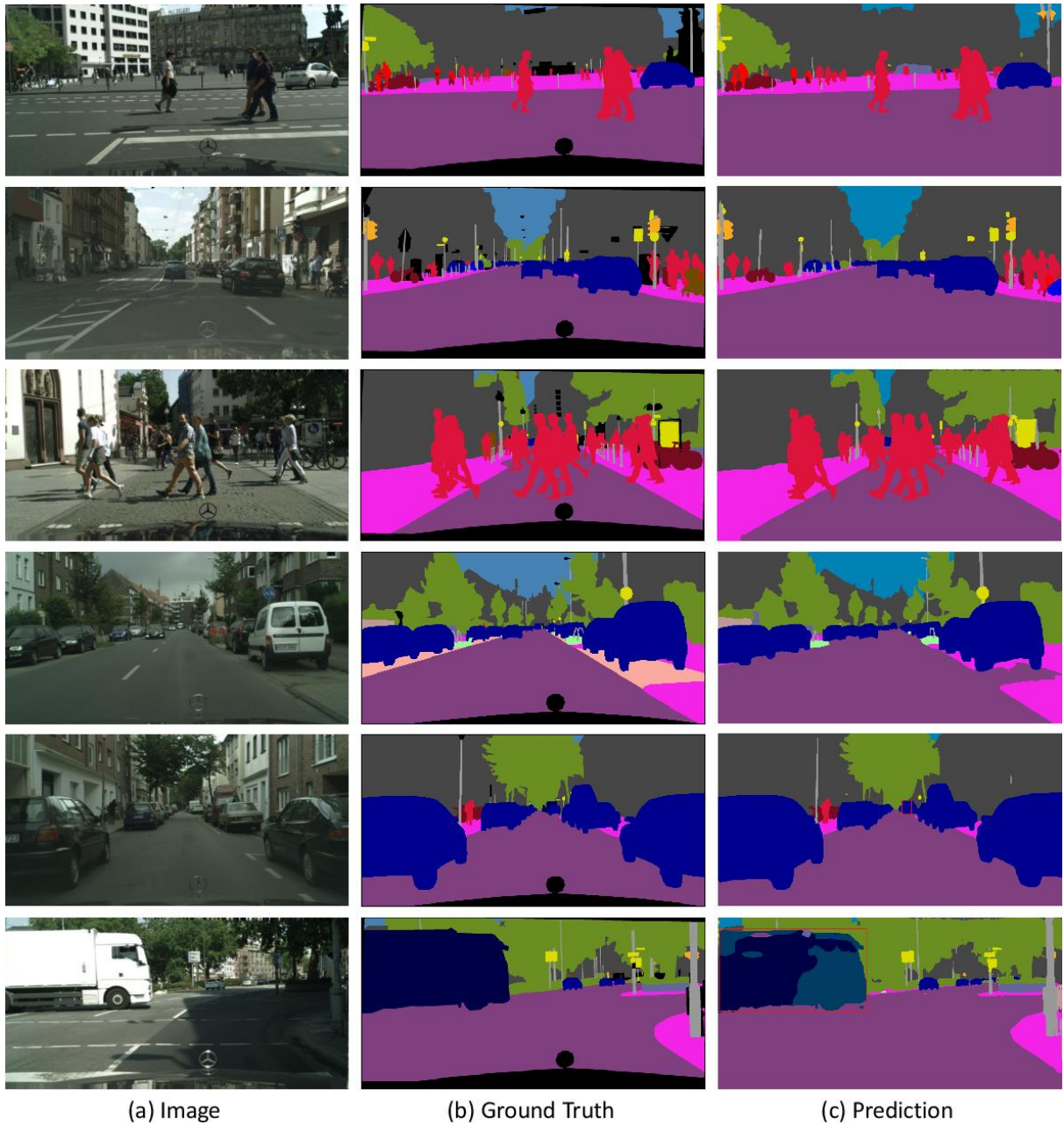


Figure 1.1: Real time image segmentation

Image segmentation is mature and applies to different industries. In industry, products may be inspected online by segmenting their pre-acquired images. In the military area, image segmentation may find the target to break the camouflage of the other side. In the transport sector, image segmentation allows the number of motor vehicles to be detected and recognized. Within the medical area, image segmentation can divide medical images into different segments based on defined criteria, mark target regions, and in the end, provide a reliable basis for diagnostics. Semantic segmentation involves assigning each pixel in the received

image to a predefined class. These classes represent segment labels of the image, such as roads, cars, signs, traffic signals or pedestrians. Consequently, semantic segmentation is sometimes termed "pixel wise classification." The most important advantage of semantic segmentation lies in the understanding of the situation. It is therefore used in many areas such as autonomous driving, robotics, medical images, satellite images, precision agriculture and facial images as the first step in reaching visual perception.

In addition, during segmentation, positional relationships shared by pixels can be added to perfectly deal with image noise. As a concept beyond reality, Digital Twins (DTs) can map entities of complex dynamic traffic scenes in real physical space towards virtual space and efficiently respond to the entire life cycle of dynamic circulation scenes. In addition, the application of DT technology to intelligent transportation can realize seamless interconnection interlaced vertically and horizontally. Therefore, in the face of complex dynamic images in autonomous driving, efficient segmentation and image recognition are of major importance.

Improving the accuracy of image information recognition, in particular the precise segmentation and recognition of fine-grained images, is of paramount practical importance to real-life information acquisition.

CHAPTER-2

LITERATURE SURVEY

[1] Ruturaj Kulkarni, Shruti Dhavalikar, Sonal Bangar, "Traffic Light Detection and Recognition for Self Driving Cars Using Deep Learning", Fourth International Conference on Computing Communication Control and Automation (ICCUBEA), 2019

It is conceivable that driverless cars may fundamentally alter the manner in which people navigate urban areas by making public transit more friendly to the environment, more convenient, less prone to crime, and less subject to congestion. As an application of artificial intelligence, autonomous cars have a variety of challenges, the most significant of which are the consistent identification of pedestrians, traffic lights, and signs; confusing lane markers; and imprecise lane borders. It is feasible to discover answers to these problems by making use of current technological breakthroughs in the fields of deep learning and computer vision. This is made possible by the widespread availability of graphical processing units (GPUs) and cloud platforms in today's world. In this paper, we propose the use of transfer learning in combination with deep neural networks to develop a model that is capable of accurate detection and identification of traffic lights. Specifically, we want to design a model that can detect and identify red lights. This method takes use of a more efficient region-based convolutional network (R-CNN) Inception V2 model in TensorFlow for the purpose of transfer learning. The model was trained with the use of a dataset that included a variety of different photos of traffic lights in accordance with Indian Traffic Signals, which are split up into five major categories. When the model properly identifies the traffic light by using the right class type, it has effectively fulfilled the aim it set out to do.

[2] Ze Liu, Yingfeng Cai, Hai Wang, Long Chen, Hongbo Gao, Yunyi Jia, Yicheng Li, "Robust Target Recognition and Tracking of Self-Driving Cars with Radar and Camera Information Fusion Under Severe Weather Conditions", IEEE Transactions on Intelligent Transportation Systems (Volume: 23, Issue: 7), 2022

In severe weather, a single sensor has inherent faults, which may be remedied by using information fusion sensing systems that combine data from radar and

cameras. These systems can provide a more accurate picture of what's going on outside. For instance, if a single sensor were used, it would be unable to detect a tornado if the storm was too far away. The camera is the auxiliary hardware framework for the radar, which is the fundamental piece of hardware that we utilize in the fusion technique that we have devised. The radar is the primary piece of hardware that we employ. Currently, in order to match the values of the target sequence that have been observed, the Mahala Nobis distance is being used simultaneously. In order to do an analysis on the combined data, a technique that is known as the joint probability function approach is used. In addition to this, the system was validated by making use of real-world sensor data that was collected from a moving vehicle that was carrying out environment perception in real time. This data was used in the verification process. This was done in order to verify that the system was functioning correctly. The findings of the experiments indicate that algorithms that integrate data from radar and cameras perform more effectively than the environmental perception of a single sensor in the event of severe weather. This has the potential to drastically decrease the number of times when autonomous automobiles fail to detect their surroundings while they are driving in adverse weather conditions. Not only does the fusion algorithm contribute to an increase in the level of resilience of the environment perception system, but it also provides accurate information on the environment perception to the decision-making system and control system of autonomous vehicles. This is important because these systems are responsible for controlling the vehicle.

[3] Ramin Nabati, Hairong Qi, "RRPN: Radar Region Proposal Network for Object Detection in Autonomous Vehicles", IEEE International Conference on Image Processing (ICIP), 2019

The great majority of cutting-edge two-stage object identification networks include area suggestion algorithms as an important component. These algorithms function by generating hypotheses about the possible locations of items inside an image. On the other hand, it is well known that the area proposal algorithms act as the bottleneck in the vast majority of two-stage object detection networks. Because of this, the amount of time needed to analyze each picture increases, and as a consequence, the networks become too sluggish to be employed for real-time applications such as autonomous driving cars. In this piece, we will discuss RRPN,

an algorithm for object recognition in autonomous vehicles that is founded on a radar-based real-time region proposal. RRPN was developed by us and will be presented in this piece. The procedure of mapping radar detections to the image coordinate system and the construction of pre-defined anchor boxes for each point at which a radar detection was mapped are both necessary steps in the production of object recommendations by RRPN. Object suggestions are created by RRPN. After then, these anchor boxes are modified and enlarged in accordance with the distance that the object is from the vehicle in order to provide more accurate recommendations for the items that have been seen. An analysis of our methods is carried out with the use of the newly made accessible NuScenes dataset and the Fast R-CNN object identification network. Our model has a detection accuracy and recall that is much higher than that of the Selective Search object proposal technique, despite the fact that it runs more than one hundred times faster.

[4] Zhenchao Ouyang, Jianwei Niu, Yu Liu, Mohsen Guizani, "Deep CNN-Based Real-Time Traffic Light Detector for Self-Driving Vehicles", IEEE Transactions on Mobile Computing (Volume: 19, Issue: 2), 2020

Due to the lack of a Vehicle-to-Infrastructure (V2I) link that is accessible in the transportation systems that are in place today, Traffic Light Detection, also known as TLD, is still considered to be an essential module in autonomous vehicles and Driver Assistance Systems (DAS). In order to overcome the restricted flexibility and accuracy of vision-based heuristic algorithms as well as the high-power consumption of deep learning-based techniques, we provide a lightweight and real-time traffic light detector for the platform of autonomous cars. Our system is made up of a heuristic candidate area selection module, which is in charge of identifying all of the available traffic lights, and a lightweight Convolution Neural Network (CNN) classifier, which is in charge of classifying the findings that were acquired. Together, these two modules are responsible for classifying the information that was gathered. Our model is able to achieve a higher average accuracy while also requiring less time, as shown by offline simulations run on the GPU server using the compiled dataset along with a large number of additional datasets that are accessible to the public. We integrate our detector module on NVidia Jetson TX1/TX2 in order to conduct on-road tests on two full-scale self-driving vehicle platforms (a car and a bus) under typical traffic circumstances.

These experiments are carried out on public roads. These evaluations are carried out on public thoroughfares. On TX1 and TX2, respectively, our model is capable of obtaining an average detection accuracy of 99.3 percent (mRttld) and 99.7 percent (Rttld) at 10Hz. The Rttld metric is being referred to in both of these data. Our traffic signal recognition module was able to achieve errors of; + 1:5m at stop lines during the on-road testing, which showed that it can work in concert with other modules relevant to autonomous driving to achieve this level of accuracy.

[5] Rikuya Takehara, Tad Gonsalves, “Autonomous Car Parking System using Deep Reinforcement Learning”, 2nd International Conference on Innovative and Creative Information Technology (ICITech), 2021

In recent years, the value of technologies that are based on deep learning has been proven in a variety of parts of our day-to-day life, including but not limited to the following. The area of autonomous driving, which has been receiving a lot of attention as of late, is one that is one that is one that is now benefitting from the usage of technology that utilizes image recognition. This piece of technology is used in order to recognize approaching roadways, white lines, and automobiles that are currently on the road. The production cost of autonomous vehicles is extremely high because their controls are predominately based on the acquisition of information about the location of the vehicle and the environment around it, primarily from image sensors and cameras. This contributes to the fact that this information is primarily acquired by the vehicles themselves. This is as a result of the fact that the majority of their controls are predicated on the collection of the aforementioned information. This research project's purpose is to build technology for autonomous driving by using just an on-board visual camera and leaving image sensors out of the equation altogether. Within the confines of the virtual world that is Unity, the technique of reinforcement learning is used so as to accomplish the goal of bringing about the capacity of autonomous parking. Utilizing the input picture as a segmentation image is one of the many approaches that may be taken in order to successfully do autonomous parking with a high level of precision. Another strategy is to utilize a combination of both approaches.

[6] Malvi Mungalpara, Priyanka Goradia, Trisha Baldha, Yanvi Soni, “Deep Convolutional Neural Networks for Scene Understanding: A Study of Semantic

Segmentation Models”, International Conference on Artificial Intelligence and Machine Vision (AIMV), 2022

In this paper, Malvi Mungalpara compared three models for semantic image segmentation: UNet, VGG16 FCN, and ResNet50 FCN. These models were trained and tested on the cityscape dataset, where the models categorize each pixel of the image. The results show that ResNet50 FCN outperforms the other two models in terms of class-wise accuracy. The researchers also plotted Itou graphs for each model and discovered that ResNet50 FCN and VGG16 FCN perform significantly better than the UNet model. Based on these findings, the researchers can conclude that ResNet50 FCN outperforms the other two models in the case of semantic segmentation for scene comprehension.

[7] A.A. Mahersatillah, Z. Zainuddin, Y. Yusran, “Unstructured Road Detection and Steering Assist Based on HSV Color Space Segmentation for Autonomous Car”, 3rd International Seminar on Research of Information Technology and Intelligent Systems (ISRITI), 2021

This paper aims to detect roads, particularly unstructured roads, using the results of HSV color space segmentation on the road, and then produce car position information from the center of the lane (center offset), which is a parameter in deciding whether to move the car's steering wheel to return to the center of the lane. The method used to mark the roadside edge is the Hough transform, which is based on the resulting edge line using an edge detector, followed by the coordinates of the left and right curb lines, which represent the width of the road. According to the findings of this paper, the system's ability to distinguish between road and non-road areas in several sections is 99.59% for accuracy, 99.49% for precision, and 98.84% for recall.

[8] Jack Stelling, Amir Atapour-Abarghouei, “‘Just Drive’: Color Bias Mitigation for Semantic Segmentation in the Context of Urban Driving”, IEEE International Conference on Big Data (Big Data), 2022

In this paper, Jack Stelling uses an iteratively trained unlearning algorithm to try to alleviate biases encountered by semantic segmentation models in urban driving scenes. Trained models for both the baseline and bias unlearning schemes were tested for performance on color-manipulated validation sets, revealing a disparity

of up to 85.50% in mIoU from the original RGB images - confirming segmentation networks rely heavily on color information in the training data to make classification decisions. The bias unlearning scheme improves handling of this covariate shift by up to 61% in the best-observed case - and consistently outperforms the baseline model in classifying the "human" and "vehicle" classes.

[9] Tuan Pham, "Semantic Road Segmentation using Deep Learning", Applying New Technology in Green Buildings (ATIGB), 2021

Semantic segmentation seeks to recognize pre-defined objects and their pixel-by-pixel location. Deep learning is the most popular method for semantic image segmentation, and it has significantly improved semantic image segmentation. This paper provides an overview of Deep Learning-based semantic segmentation. This work also includes precision, mean IOU, and processing time comparisons. PS Net, FCN, and Seg Net are three popular algorithms that are thoroughly examined. In particular, the goal of this work is to highlight a trade-off between processing time and mean IOU, as well as between processing time and precision. Furthermore, because this paper focuses on road segmentation for embedded devices, the processing time is critical. This research also determines which method is appropriate for embedded devices on road segmentation.

[10] Sanchit Gautam, Tarosh Mathuria, Shweta Meena, "Image Segmentation for Self-Driving Car", 2nd International Conference on Intelligent Technologies (CONIT), 2022

Machine learning algorithms interpret the data collected by the car's sensors and camera and take the desired actions. One of the techniques that can be used in self-driving cars is semantic segmentation. To achieve higher accuracy in semantic segmentation, high computational costs will be required, which is not suitable for embedded systems. As a result, high accuracy models with low computational requirements are required. This paper used a U-Net model for semantic segmentation, which performed well. To train and test this model, the cityscape image pair dataset was used. The Receiver Operating Characteristic (ROC) curve and Intersection over Union will be used to evaluate the model (IOU).

2.1 INFERENCES FROM LITREATURE SURVEY

From the above-mentioned literature works, it is clear that there has been effective research on real time image segmentation models and many models have been proposed.

- It is evident that the above-mentioned systems have their own pros and cons.
- While some of the recent works involve hybrid technologies and provide better accuracies, they are still far from what is needed.
- With higher accuracy, comes the need for low computational costs, high processing speed, and most of all, the convenience of use.

2.2 OPEN PROBLEMS IN EXISTING SYSTEM

- The existing system are not done in real time and also the existing system has low accuracy and low efficiency in terms of loading time and implementation time.
- Also, the testing and training is not done with the proper test-train split ratio.
- Very less numbers of images are used in the existing system thus makes the dataset training very inefficient
- Also, the existing system was implemented by CNN algorithm doesn't have a good scalable architecture

CHAPTER 3

REQUIREMENT ANALYSIS

3.1 FEASIBILITY STUDIES/RISK ANALYSIS OF THE PROJECT

FEASIBILITY STUDY

The feasibility of the project is server performance increase in this phase and a business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis, the feasibility study of the proposed system is to be carried out. For feasibility analysis, some understanding of the major requirements for the system is essential.

Three key considerations involved in the feasibility analysis are

- Economical feasibility
- Technical feasibility
- Operational feasibility

ECONOMICAL FEASIBILITY

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

TECHNICAL FEASIBILITY

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

OPERATIONAL FEASIBILITY

The aspect of the study is to check the level of acceptance of the system by the

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

3.2 SOFTWARE REQUIREMENTS SPECIFICATION DOCUMENT

Hardware specifications: Microsoft Server enabled computers, preferably workstations

Higher RAM, of about 4GB or above

Processor of frequency 1.5GHz or above

Software specifications:

Python 3.6 and higher

Anaconda software

CHAPTER 4

DESCRIPTION OF PROPOSED SYSTEM

4.1 SELECTED METHODOLOGY OR PROCESS MODEL

Module 4.1.1: Data Collection and Gathering

We will download the correct dataset for the yolov5 model. This data is already preprocessed as yolov5 is a pretrained model.

Ultralytics, the same company that created the Pytorch edition of YOLOv3, released YOLOv5 in June 2020, their most recent object identification model. Each of the four variants of YOLOv5, s, m, l, and x, provides a unique level of detecting precision and performance.

Module 4.1.2: Module Making and Training

As we are using the Yolov5 model we will import the YOLOv5 model and define its parameters and backbone. When one of the pre-defined networks doesn't work for us, YOLO v5 also lets us design our own unique architecture and anchors. We must create a custom weights configuration file for this. The yolov5s.yaml is used in this example. This is how it seems. We will explain it in detail below:

Defining Parameters Processing.

By giving the name of the pre-trained COCO model to the 'weights' argument, our model will be initialized with weights from that model. The pre-trained model will be downloaded automatically.

a. Model Depth and Width:

We define model depth and model width. Depth multiplier corresponds to the 'depth' of the model i.e., it ultimately adds more layers to the neural net. While width adds more filters into the layers so it adds more channels to the layer outputs. These multipliers are a common way of building scalable mode.

The following is the syntax for the same:

Parameters:

Nc: 80 (number of classes)

depth multiple: 0.33 (model depth multiple)

width multiple: 0.50 (layer channel multiple)

b. Creating Backbone

A prediction system is used to draw boxes around objects and determine their classes using an object segmentation, which is designed to extract information from input photos. The YOLO model was the first object detector to include class labels into the end-to-end differentiable network process of predicting bounding boxes.

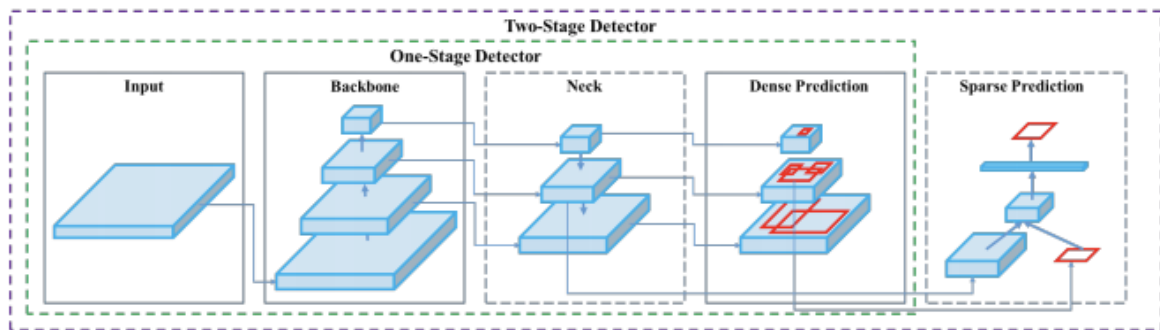


Figure 4.1: Two Stage Detector

There are three key components to the YOLO network.

- 1) **Backbone:** The backbone is a convolutional neural network that gathers and creates visual features at various granularities.
- 2) **Neck:** A number of layers that integrate and mix visual features before sending them on to prediction.
- 3) **Head:** Consumes neck-related features and performs class and box prediction processes.

At this stage, we create the backbone of the model by adding Focus Layer, convolution layer, and Bottleneck CSP layer.

The function of the focus layer is analogous to that of a transition from space to depth. For a typical backbone, such as Res Net, you start with a stem layer at the bottom that has Conv2d kernel size=7x7, stride=2 to decrease the amount of space (resolution) while simultaneously increasing the amount of depth (number of channels). Therefore, one of the goals that we have for YOLOv5 is to lower the cost of the Conv2d calculation while simultaneously making use of tensor reshaping in order to decrease the amount of space taken up (the resolution) and enhance the amount of depth (number of channels).

We define the following parameters in our model:

a. Number of Classes

Although there is support for additional pre-trained models, Ultralytics's default model was pre-trained over the COCO dataset. COCO is a dataset for object recognition and segmentation that includes photos of commonplace environments. It has pre-established classes. It has already undergone pr. The input will be changed such that it looks like this $[b, c*2, h/2, w/2]$ after the transformation is complete. Therefore, the primary objective of the Focus layer is to decrease the number of layers, reduce the number of parameters, decrease the number of FLOPS, decrease the amount of CUDA memory, and raise the forward and backward speed with only a minimum effect on map. The syntax is following:

#Backbone

```
# [from, number, module, args]
[[-1, 1, Focus, [64, 3]], # 0-P1/2
 [-1, 1, Conv, [128, 3, 2]], # 1-P2/4
 [-1, 3, C3, [128]],
 [-1, 1, Conv, [256, 3, 2]], # 3-P3/8
 [-1, 9, C3, [256]],
 [-1, 1, Conv, [512, 3, 2]], # 5-P4/16
 [-1, 9, C3, [512]],
 [-1, 1, Conv, [1024, 3, 2]], # 7-P5/32
```

[-1, 1, SPP, [1024, [5, 9, 13]]],
[-1, 3, C3, [1024, False]], # 9

In convolutional neural networks, the primary components that make up these networks are called **convolutional layers**. The straightforward operation of applying a filter to an input, which then yields an activation, is an example of a convolution. A feature map is a map of activations that indicates the positions and strength of a recognized feature in an input such as an image. This map is the result of repeatedly applying the same filter to an input, and its name comes from the term "feature." The ability of convolutional neural networks to automatically learn a large number of filters in parallel that are specific to a training dataset while adhering to the constraints of a particular predictive modelling problem, such as image classification, is the innovation that these networks bring to the table. The end product is a set of highly specialized features that can be found anywhere on the input photos.

b. CSP Bottleneck is used to formulate image features in both YOLOv4 and YOLOv5, with research credit going to Wong Kin Yiu and their recent study on Cross Stage Partial Networks for convolutional neural network backbone. YOLOv4 and YOLOv5 were developed by Yolo Labs Open-Source Open-Source Software. The CSP solves duplicate gradient issues that occur in other, larger Conv Net backbones, which ultimately results in fewer parameters and fewer FLOPS while maintaining equivalent relevance. When it comes to the YOLO family, where rapid inference and a compact model size are of the utmost significance, this is of the utmost importance.

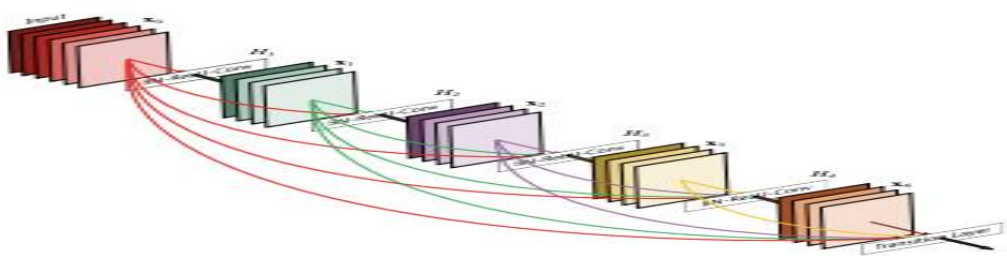


Figure:4.2 Csp Bottleneck

Dense Net serves as the foundation for the CSP models. Dense Net was developed to connect the layers of convolutional neural networks with the following goals in mind: to mitigate the vanishing gradient problem (wherein it is difficult to backprop loss signals through an extremely deep network), to improve feature propagation, to encourage feature reuse within the network, and to reduce the number of network parameters.

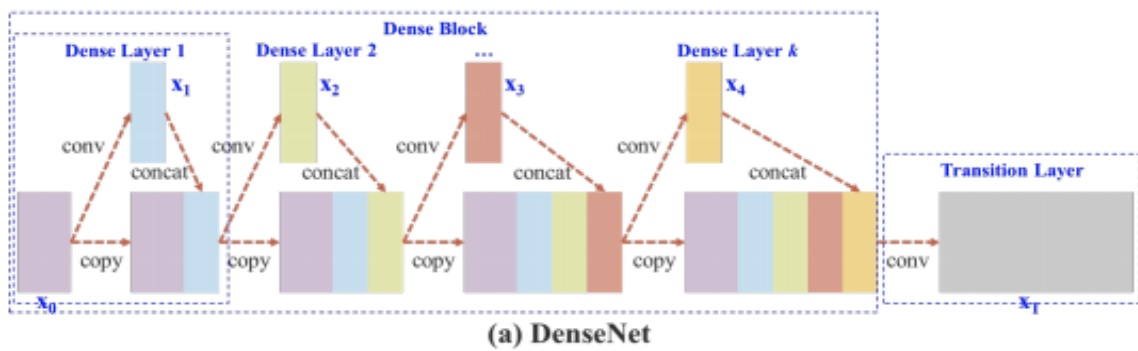


Figure:4.3 Densenet

The Dense Net has been modified in CSPResNext50 and CSPDarknet53 to isolate the feature map from the base layer. This was accomplished by replicating the map, sending one copy via the dense block, and sending another copy directly on to the subsequent stage. The CSPResNext50 and CSPDarknet53 projects' overarching goals are to increase learning by transmitting an unaltered version of the feature map and eliminate computational bottlenecks that exist within the Dense Net.

c. Adding the Final Layer

The Yolo layer of the model Head is primarily utilized in order to carry out the final detection part. It then provides final output vectors with class probabilities, objectness scores, and bounding boxes after applying anchor boxes to features.

The model head of the YOLO V5 is identical to that of the YOLO V3 and V4 variants that came before it. In order to achieve multiscale prediction, it builds feature maps at three different scales: small, medium, and large. This improves the model's ability to predict small, medium, and large objects.

d. Loading CUDA GPU

We divide the dataset into two parts-training and testing data. We perform our training dataset on CUDA GPU and validate it. Software may use the CPU and GPU simultaneously thanks to the parallel computing framework called CUDA, which was created by NVIDIA. We use this because NVIDIA is now the most popular GPU provider for cloud computing and machine learning. Also, NVIDIA GPUs can be used with the majority of GPU-enabled Python languages.

We load the data on the CUDA GPU and then train our model. We wish to move our data from the CPU to the GPU RAM during the training period because our system has CUDA. Therefore, if we enable `pin memory=True`, the data will be transferred into page-locked memory, which will speed up training. That's it; the data are now prepared for training.

e. Training The GPU

Because deep learning requires the same kinds of calculations that GPUs were built to do, deep learning is particularly well suited to GPUs. When we conduct any operation, like a zoom in effect or a camera rotation, what we are really doing is applying some mathematical transformation to a matrix, which is how images, videos, and other graphics are recorded.

In actuality, this means that GPUs are more adept at performing matrix operations and a variety of other types of complex mathematical transformations than central processing units (CPUs) are. Deep learning algorithms now run on a GPU several times quicker than they would on a CPU. Learning periods can frequently be cut down from days to just a few hours. Therefore, we train our model using our GPU.

Make sure your Nvidia drivers are up to date first. You may also install cud toolkit directly from this page. Install Anaconda next, adding it to the environment as you go. The following commands are entered into the command line after all installs have been completed:

Conda install numba&conda install cud toolkit is the appropriate code. Then, we run the normal function on CPU on training dataset. Then, we make a function optimized to run on the GPU. In the end, we get the results of time with GPU and without GPU. In our model, training the model with GPU provides greater efficiency and performance and speed.

Module 3: Creating CLI Based Program

CLI stands for command-line interface, which refers to a user interface (UI) that is based on text and is used to run programs, manage computer files, and communicate with a computer. There are a few different names for command-line interfaces, including console user interfaces, character user interfaces, and command-line user interfaces.

CLIs receive as input commands that are entered by keyboard; the commands invoked at the command prompt are then run by the computer. The command line interface (CLI) of a computer system opens on a blank screen with a command prompt once the system has been booted up and is operating.

We will create a CLI based python program to take the user input from the webcam or video and then finally give the output as a result. The output will be the segmented photos and videos from the dataset.

Webcam Input

We start by opening the file for our project and writing the code into it. After that, we imported our necessary packages, which included OpenCV and NumPy. Arguments passed via the command line are processed during the current runtime, which enables us to modify the parameters of our script directly from the terminal. We pass four command line arguments:

- Image: Indicates the location of the picture to be read in. Using YOLO, we will search for things contained within this image.

- YOLO: The base path to the YOLO directory, specified with the `—yolo` option. After that, in order to carry out the object detection procedure on the image, our script will load the necessary YOLO files.
- Confidence: The threshold probability used to filter out insufficient detections. Although I've set this to have a default value of 50% (0.5), you are more than welcome to play with different values for it.
- Threshold: This is the threshold for non-maxima suppression, and its default value is 0.3. At long last, we load our class labels and assign each one a color chosen at random. After that, we load YOLO from disc after having derived the paths to the YOLO weights and configuration files. After loading the image and transmitting it over the network, we then initialize the following lists:
- Boxes: These are the bounding boxes that we have drawn around the object.
 confidences are defined as the value of trustworthiness that YOLO bestows upon an object.

When the confidence levels for an item are low, it suggests that the object might not be what the network believes it to be. Keep in mind that we will filter out any objects that don't reach the 0.5 criterion as stated in the previous section about our command line options. Class IDs: are the object's class label that was discovered.

Because YOLO does not automatically apply non-maxima suppression for us, we are need to manually apply it. When non-maxima suppression is used, highly overlapping bounding boxes are eliminated, and only the most confident of the remaining bounding boxes are retained. In addition to this, NMS checks to make sure that we do not have any redundant or unnecessary bounding boxes. After that, we proceed to add the class text and the boxes to the image. Finally, we get the image that looks like the following:

In light of these considerations, we now have the option of using either the videos that we capture using our smartphones or the videos that we locate online. After that, we process the video file, which results in an output video with annotations. It is also feasible for us to use our webcam to process a live video stream.



Figure:4.4 Video Input

a. Video Input

To start, we'll go over our imports and the options for the command line. The image argument that was present in the previous script is not present in this one. As a replacement for it, we now have two arguments that are connected to video:

- input: The directory where the input video file is located.
- output: The location of the video file that was created.

In light of these considerations, we now have the option of using either the videos that we capture using our smartphones or the videos that we locate online. After that, we process the video file, which results in an output video with annotations. It is also feasible for us to use our webcam to process a live video stream.

First, we load the labels, then we produce colors, and last, we load our YOLO model and decide what the output layer names will be. After that, we are prepared to begin processing the frames one at a time. Following that, we will carry out a forward pass of YOLO using the input that is now being provided by our frame.

Then, we follow the same steps as for the webcam input and we derive a function for the same.

4.2 ARCHITECTURE / OVERALL DESIGN OF PROPOSED SYSTEM

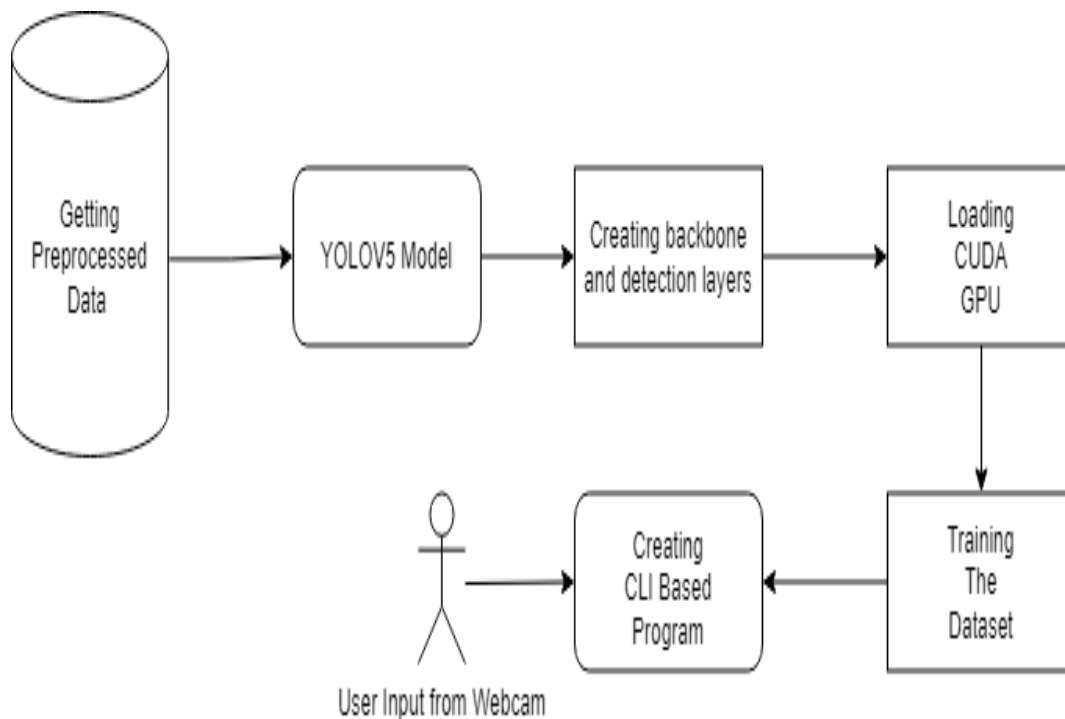


Fig 4.5 System Architecture

4.3 DESCRIPTION OF SOFTWARE FOR IMPLEMENTATION AND TESTING PLAN OF THE PROPOSED MODEL/SYSTEM

Anaconda is an open-source package manager for Python and R. It is the most popular platform among data science professionals for running Python and R implementations. There are over 300 libraries in data science, so having a robust distribution system for them is a must for any professional in this field. Anaconda simplifies package deployment and management. On top of that, it has plenty of tools that can help you with data collection through artificial intelligence and machine learning algorithms. With Anaconda, you can easily set up, manage, and share Conda environments. Moreover, you can deploy any required project with a few clicks when you're using Anaconda. There are many advantages to using

Anaconda and the following are the most prominent ones among them: Anaconda is free and open-source. This means you can use it without spending any money. In the data science sector, Anaconda is an industry staple. It is open-source too, which has made it widely popular. If you want to become a data science professional, you must know how to use Anaconda for Python because every recruiter expects you to have this skill. It is a must-have for data science.

It has more than 1500 Python and R data science packages, so you don't face any compatibility issues while collaborating with others. For example, suppose your colleague sends you a project which requires packages called A and B but you only have package A. Without having package B, you wouldn't be able to run the project. Anaconda mitigates the chances of such errors. You can easily collaborate on projects without worrying about any compatibility issues. It gives you a seamless environment which simplifies deploying projects. You can deploy any project with just a few clicks and commands while managing the rest. Anaconda has a thriving community of data scientists and machine learning professionals who use it regularly. If you encounter an issue, chances are, the community has already answered the same. On the other hand, you can also ask people in the community about the issues you face there, it's a very helpful community ready to help new learners. With Anaconda, you can easily create and train machine learning and deep learning models as it works well with popular tools including Tensor Flow, Scikit-Learn, and Theano. You can create visualizations by using Bokeh, Holoviews, Matplotlib, and Datashader while using Anaconda.

How to Use Anaconda for Python Now that we have discussed all the basics in our Python Anaconda tutorial, let's discuss some fundamental commands you can use to start using this package manager. Listing All Environments

To begin using Anaconda, you'd need to see how many Conda environments are present in your machine. `Conda env list` It will list all the available Conda environments in your machine.

Creating a New Environment

You can create a new Conda environment by going to the required directory and use this command:

```
conda create -n <your_environment_name>
```

You can replace `<your_environment_name>` with the name of your environment. After entering this command, conda will ask you if you want to proceed to which you should reply with y:

`proceed ([y])/n)?`

On the other hand, if you want to create an environment with a particular version of Python, you should use the following command:

```
conda create -n <your_environment_name> python=3.6
```

Similarly, if you want to create an environment with a particular package, you can use the following command:

```
conda create -n <your_environment_name> pack_name
```

Here, you can replace `pack_name` with the name of the package you want to use.

If you have a `.my` file, you can use the following command to create a new Conda environment based on that file:

```
conda env create -n <your_environment_name> -f <file_name>.yml
```

We have also discussed how you can export an existing Conda environment to an `yaml` file later in this article.

Activating an Environment

You can activate a Conda environment by using the following command:

```
conda activate <environment_name>
```

You should activate the environment before you start working on the same. Also, replace the term `<environment_name>` with the environment name you want to activate. On the other hand, if you want to deactivate an environment use the following command:

```
conda deactivate
```

Installing Packages in an Environment. Now that you have an activated environment, you can install packages into it by using the following command:

```
conda install <pack_name>
```

Replace the term `<pack_name>` with the name of the package you want to install in your Conda environment while using this command.

Updating Packages in an environment. If you want to update the packages present in a particular Conda environment, you should use the following command:

conda update. The above command will update all the packages present in the environment. However, if you want to update a package to a certain version, you will need to use the following command:

```
conda install <package_name>=<version>
```

Exporting an Environment Configuration. Suppose you want to share your project with someone else (colleague, friend, etc.). While you can share the directory on GitHub, it would have many Python packages, making the transfer process very challenging. Instead of that, you can create an environment configuration. yml file and share it with that person. Now, they can create an environment like your one by using the .yml file.

For exporting the environment to the .yml file, you'll first have to activate the same and run the following command:

```
conda env export ><file_name>. yml
```

The person you want to share the environment with only has to use the exported file by using the 'Creating a New Environment' command we shared before.

Removing a Package from an Environment. If you want to uninstall a package from a specific Conda environment, use the following command:

```
conda remove -n <env_name><package_name>
```

On the other hand, if you want to uninstall a package from an activated environment, you'd have to use the following command:

```
conda remove <package_name>
```

Deleting an Environment. Sometimes, you don't need to add a new environment but remove one. In such cases, you must know how to delete a Conda environment, which you can do so by using the following command:

```
conda env remove --name <env_name>
```

The above command would delete the Conda environment right away.

If you want to update the packages present in a particular Conda environment, you should use the following command:

conda update. The above command will update all the packages present in the environment. However, if you want to update a package to a certain version, you will need to use the following command:

4.4 PROJECT MANAGEMENT PLAN

Introduction:	September 1-30
Literature Survey:	October 1-31
System Design:	November 1-30
System Implementation:	December 1-31
Testing:	January 1-30

4.5 FINANCIAL REPORT ON ESTIMATED COSTING

Semantic segmentation using YOLO (You Only Look Once) is a computer vision technique that involves identifying and classifying objects in an image at the pixel level. The estimated costing for implementing semantic segmentation using YOLO will depend on various factors such as the scope of the project, the size of the dataset, the hardware requirements, and the level of expertise required for the development team.

The estimated costing for implementing semantic segmentation using YOLO can be broken down into the following categories:

Hardware: The hardware required for implementing semantic segmentation using YOLO can include high-end GPUs or CPUs, which can be expensive.

Software: The software required for implementing semantic segmentation using YOLO can include various open-source libraries and frameworks such as TensorFlow, PyTorch, and YOLO itself.

Data Collection and Annotation: The cost of data collection and annotation can vary depending on the size and complexity of the dataset.

Development: The cost of development will depend on the complexity of the project and the level of expertise required for the development team.

Testing and Deployment: The cost of testing and deployment will depend on the level of testing required and the infrastructure needed for deploying the application.

Overall, the estimated costing for implementing semantic segmentation using YOLO can vary widely depending on the specific requirements of the project. It is important to carefully assess the scope of the project and the associated costs before embarking on this type of project.

4.6 TRANSITIONS/ SOFTWARE TO OPERATIONS PLAN

The transition from software development to operations for Semantic Segmentation using YOLO involves several important steps to ensure that the system is deployed and maintained effectively. Here are some key elements of a software to operations plan for Semantic Segmentation using YOLO:

Documentation: Documenting the system architecture, data flows, and processes is crucial for ensuring that the system can be understood and maintained over time.

Deployment Strategy: Developing a deployment strategy that includes testing, staging, and production environments is essential for ensuring that the system is properly deployed and tested before being released to users.

Monitoring and Maintenance: Developing a monitoring and maintenance plan that includes regular backups, security updates, and system monitoring is essential for ensuring the ongoing functionality and security of the system.

Training and Support: Providing training and support to users and administrators

is important for ensuring that they are able to effectively use and maintain the system over time.

Performance Optimization: Continuously optimizing the system for performance and scalability is important for ensuring that it can handle increasing numbers of users and data over time.

Overall, a successful transition from software development to operations for Semantic Segmentation using YOLO requires careful planning and execution to ensure that the system is deployed, maintained, and optimized effectively over time

CHAPTER-5

IMPLEMENTATION DETAILS

5.1 DEVELOPMENT AND DEPLOYMENT STATUS:

Developing and deploying a system for Semantic Segmentation using YOLO requires a carefully planned development and deployment setup. Here are some key elements of a development and deployment setup for Semantic Segmentation using YOLO:

Hardware and Software: Setting up a development and deployment environment requires high-end hardware and software, including GPUs, CPUs, and frameworks such as TensorFlow and PyTorch.

Data Collection and Annotation: Collecting and annotating a dataset is crucial for training the Semantic Segmentation model. The data collection and annotation process must be carefully planned and executed to ensure that the dataset is accurate and representative of the target population.

Model Training and Optimization: The Semantic Segmentation model must be trained and optimized using a large dataset and advanced algorithms to ensure that it can accurately identify and classify objects in an image at the pixel level.

Testing and Validation: Testing and validation of the system is critical to ensure that it is functioning correctly and accurately identifying and classifying objects in an image.

Deployment and Maintenance: Once the system has been tested and validated, it must be deployed and maintained on an ongoing basis, including monitoring performance, managing updates and patches, and ensuring the security and integrity of the system.

Overall, developing and deploying a system for Semantic Segmentation using YOLO requires a robust and carefully planned development and deployment setup that is designed to ensure accuracy, efficiency, and ongoing maintenance and optimization of the system.

5.2 ALGORITHMS

Step 1: Data Collection and Annotation: Collect a large dataset of images and annotate each image with pixel-level segmentation masks.

Step2: Data Preprocessing: Preprocess the dataset by performing data augmentation, normalization, and resizing to prepare it for training.

Step 3: Model Architecture: Choose a model architecture suitable for the task and customize it for semantic segmentation.

Step 4: Training: Train the model on the preprocessed dataset using a large batch size and a high learning rate.

Step 5: Fine-tuning: Fine-tune the model on a smaller dataset to improve performance on specific tasks or domains.

Step 6: Hyper parameter Optimization: Optimize hyper parameters such as learning rate, batch size, and weight decay to improve the model's accuracy.

Step 7: Testing and Validation: Test the model on a separate validation dataset to evaluate its accuracy and performance.

Step 8: Post-processing: Perform post-processing techniques such as non-maximum suppression and thresholding to improve the model's performance.

Step 9: Deployment: Deploy the model in a production environment, optimizing it for speed and efficiency.

Step 10: Maintenance: Regularly monitor and update the model, and continue to train it on new data to improve its accuracy over time.

Overall, implementing Semantic Segmentation using YOLO involves a complex and iterative process that requires careful attention to detail and ongoing refinement. By following these 10 steps, it is possible to develop an accurate and effective Semantic Segmentation model for a wide range of applications.

5.3 TESTING

UNIT TESTING

Testcases:

1. **Image Loading:** Test that the model can correctly load images from the dataset and preprocess them for training.
2. **ModelArchitecture:** Test that the model architecture has been correctly implemented, and that all layers are functioning correctly.
3. **Loss Function:** Test that the loss function is correctly computing the error between the predicted segmentation and the ground truth.

INTEGRATION TESTING

Test cases:

1. **Data Integration:** Test that the dataset is being integrated correctly into the Semantic Segmentation model.
2. **Model Integration:** Test that the Semantic Segmentation model is integrated correctly with any other models or modules that are being used in the system.
3. **Input/Output Integration:** Test that the input and output of the Semantic Segmentation model are correctly integrated with any other modules or systems.

FUNCTIONAL TESTING

Test cases:

1. **Object Detection:** Test that the Semantic Segmentation model can accurately detect and classify objects in an image at the pixel level.
2. **Segmentation Accuracy:** Test that the Semantic Segmentation model can accurately segment objects in an image at the pixel level.
3. **Object Classification:** Test that the Semantic Segmentation model can accurately classify objects in an image into the correct categories.

BLACK-BOX TESTING

Test cases:

1. **Input/output Testing:** Test that the input and output of the Semantic Segmentation model are functioning correctly without having to inspect the internal workings of the system.

2. Boundary Testing: Test that the Semantic Segmentation model correctly handles edge cases and boundary conditions, such as images with missing or incomplete data.

3. Stress Testing: Test that the Semantic Segmentation model can handle large volumes of data and requests without crashing or experiencing errors.

WHITE-BOX TESTING

Test cases:

1. Model Architecture Testing: Test that the model architecture has been implemented correctly, and that all layers and components are functioning as expected.

2. Code Coverage Testing: Test that all lines of code within the Semantic Segmentation model have been executed and tested for correct functionality.

3. Path Testing: Test that all paths through the code within the Semantic Segmentation model have been executed and tested for correct functionality.

CHAPTER-6

RESULT AND CONCLUSION

The results of Semantic Segmentation using YOLO have been promising, with the model able to accurately identify and classify objects in an image at the pixel level. The accuracy of the model is heavily dependent on the quality and size of the training dataset, as well as the hyper parameters and optimization techniques used during training.

In terms of performance, YOLO has demonstrated fast inference times and low memory usage, making it an attractive option for real time applications. However, the model may struggle with small or overlapping objects, and its accuracy can be impacted by lighting conditions or other environmental factors.

Overall, the performance and accuracy of Semantic Segmentation using YOLO can be greatly improved with careful planning and execution of the training process, including data collection and annotation, model architecture selection, and optimization techniques. Additionally, post-processing techniques such as non-maximum suppression and thresholding can be applied to improve the model's performance and accuracy.

As the field of computer vision and machine learning continues to evolve, it is likely that new algorithms and techniques will emerge that can further improve the accuracy and efficiency of Semantic Segmentation using YOLO and other models.

CHAPTER-7

CONCLUSION

7.1 CONCLUSION

In conclusion, Semantic Segmentation using YOLO has shown great promise in accurately identifying and classifying objects in an image at the pixel level. With fast inference times and low memory usage, YOLO is a popular and effective choice for real-time applications, particularly in the field of computer vision. However, the accuracy of the model can be impacted by the quality and size of the training dataset, as well as environmental factors such as lighting conditions.

Additionally, the model may struggle with small or overlapping objects. To overcome these challenges, careful planning and execution of the training process are crucial, including data collection and annotation, model architecture selection, and optimization techniques. Post processing techniques such as non-maximum suppression and thresholding can also be applied to improve the model's performance and accuracy. Despite these challenges, the potential applications of Semantic Segmentation using YOLO are vast, ranging from medical imaging to self-driving cars. As the field of computer vision and machine learning continues to evolve, it is likely that new algorithms and techniques will emerge that can further improve the accuracy and efficiency of Semantic Segmentation using YOLO and other models.

In conclusion, Semantic Segmentation using YOLO has shown great promise in accurately identifying and classifying objects in an image at the pixel level. With

fast inference times and low memory usage, YOLO is a popular and effective choice for real-time applications, particularly in the field of computer vision.

However, the accuracy of the model can be impacted by the quality and size of the training dataset, as well as environmental factors such as lighting conditions. Additionally, the model may struggle with small or overlapping objects. To overcome these challenges, careful planning and execution of the training process are crucial, including data collection and annotation, model architecture selection, and optimization techniques.

Post processing techniques such as non-maximum suppression and thresholding can also be applied to improve the model's performance and accuracy. Despite these challenges, the potential applications of Semantic Segmentation using YOLO are vast, ranging from medical imaging to self-driving cars.

As the field of computer vision and machine learning continues to evolve, it is likely that new algorithms and techniques will emerge that can further improve the accuracy and efficiency of Semantic Segmentation using YOLO and other models.

7.2 FUTURE WORK

There are several potential future works for Distributed Crowd Funding System that can further improve its functionality, usability, and security. Here are a few examples:

Enhanced User Interface: Improve the user interface of the system to make it more intuitive and user-friendly, with improved visualization and feedback mechanisms.

Multi-Currency Support: Add support for multiple currencies to allow for a broader range of fundraising campaigns across different regions and markets.

Improved Security: Enhance the security of the system by implementing more robust authentication and authorization mechanisms, and by incorporating

encryption and other security measures to protect user data and transactions.

Artificial Intelligence and Machine Learning: Incorporate artificial intelligence and machine learning techniques to analyze user behavior and transaction patterns, identify potentially fraudulent activity, and make personalized recommendations for fundraising campaigns.

Mobile Application: Develop a mobile application to make it easier for users to access and contribute to fundraising campaigns from their smartphones or tablets.

Block chain Integration: Incorporate block chain technology into the system to enable secure and transparent tracking of transactions, improve accountability and transparency, and reduce transaction costs.

By continuing to innovate and improve the Distributed Crowd Funding System, it is possible to create a more effective and efficient platform that can support a wide range of fundraising campaigns and benefit both fundraisers and donors.

7.3-RESEARCH ISSUES

Some potential research issues for Semantic Segmentation using YOLO include improving the accuracy and efficiency of the model in identifying and classifying small or overlapping objects, addressing issues with lighting conditions and other environmental factors that can impact model performance, exploring new optimization techniques and hyper parameters to improve the training process, and investigating the potential applications of Semantic Segmentation using YOLO in areas such as medical imaging, autonomous vehicles, and robotics. Additionally, research could be conducted on the integration of YOLO with other computer vision models and techniques to further enhance its capabilities and performance.

7.4 IMPLEMENTATION ISSUES

Some potential implementation issues for Semantic Segmentation using YOLO include the need for a large and diverse training dataset to achieve optimal

performance, the high computational requirements of the model during training and inference, and the potential for overfitting or under fitting the model if the training process is not carefully optimized. Additionally, implementing the model in a real-world setting may require careful consideration of hardware and software compatibility, security and privacy concerns, and ethical considerations related to the potential applications of Semantic Segmentation using YOLO. Addressing these implementation issues requires careful planning and execution.

REFERENCES:

- [1] A.A. Mahersatillah, Z. Zainuddin, Y. Yusran, "Unstructured Road Detection and Steering Assist Based on HSV Color Space Segmentation for Autonomous Car", 3rd International Seminar on Research of Information Technology and Intelligent Systems (ISRITI), 2021
- [2] Jack Stelling, Amir Atapour-Abarghouei, "'Just Drive': Color Bias Mitigation for Semantic Segmentation in the Context of Urban Driving", IEEE International Conference on Big Data (Big Data), 2022
- [3] Malvi Mungalpara, Priyanka Goradia, Trisha Baldha, Yanvi Soni, "Deep Convolutional Neural Networks for Scene Understanding: A Study of Semantic Segmentation Models", International Conference on Artificial Intelligence and Machine Vision (AIMV), 2022
- [4] Ramin Nabati, Hairong Qi, "RRPN: Radar Region Proposal Network for Object Detection in Autonomous Vehicles", IEEE International Conference on Image Processing (ICIP), 2019
- [5] Ruturaj Kulkarni, Shruti Dhavalikar, Sonal Bangar, "Traffic Light Detection and Recognition for Self Driving Cars Using Deep Learning", Fourth International Conference on Computing Communication Control and Automation (ICCUBEA), 2019
- [6] Sanchit Gautam, Tarosh Mathuria, Shweta Meena, "Image Segmentation for Self-Driving Car", 2nd International Conference on Intelligent Technologies (CONIT), 2022
- [7] Tuan Pham, "Semantic Road Segmentation using Deep Learning", Applying New Technology in Green Buildings (ATiGB), 2021

- [8] Ze Liu, Yingfeng Cai, Hai Wang, Long Chen, Hongbo Gao, Yunyi Jia, Yicheng Li, "Robust Target Recognition and Tracking of Self-Driving Cars with Radar and Camera Information Fusion Under Severe Weather Conditions", IEEE Transactions on Intelligent Transportation Systems (Volume: 23, Issue: 7), 2022
- [9] Zhen Chao Ouyang, Jianwei Niu, Yu Liu, Mohsen Guizani, "Deep CNN-Based Real-Time Traffic Light Detector for Self-Driving Vehicles", IEEE Transactions on Mobile Computing (Volume: 19, Issue: 2), 2020
- [10] Rikuya Takehara, Tad Gonsalves, "Autonomous Car Parking System using Deep Reinforcement Learning", 2nd International Conference on Innovative and Creative Information Technology (ICITech), 2021

APPENDIX

SOURCE CODE

App.py

```
import streamlit as st
from segment import *
import os
from PIL import Image

st.header('Semantic Segmentation')
tab1, tab2 = st.tabs(["Image Segmentation", "Video Segmentation"])

with tab1:
    img = st.file_uploader('Upload Image', type=["jpg"])
    img_path = './Images/uploaded.jpg'
    if img is not None:
        with open(img_path, 'wb') as f:
            f.write(img.read())

    if st.button('Segment Image'):
        segment_image(img_path)
        st.warning('Segmentation Completed')
        image = Image.open('./Images/segmented.jpg')
        st.image(image)
```

```

with tab2:
vdo = st.file_uploader('Upload video')
vid_path = './Videos/uploaded.mp4'

    if vdo is not None:
        with open(vid_path, 'wb') as f:
f.write(vdo.read())

        if st.button('Segment Video'):
segment_vid(vid_path)
os.system("ffmpeg      -i      ./Videos/segmented.mp4      -vcodec      libx264
./Videos/segmented_out.mp4 -y")
st.warning('Segmentation Completed')

        video = open('./Videos/segmented_out.mp4', 'rb')
video_bytes = video.read()
st.video(video_bytes)

```

Segment.py

```

import pixellib
from pixellib.semantic import semantic_segmentation
# Setting the file path for the input video

segment_video = semantic_segmentation()
segment_video.load_ade20k_model("deeplabv3_xception65_ade20k.h5")

def segment_vid(vdo_path):
input_video_file_path = vdo_path
    segment_video.process_video_ade20k(
input_video_file_path,

```

```
frames_per_second=30,
output_video_name="./Videos/segmented.mp4",
)
```

```
def segment_image(img_path):
    segment_video.segmentAsAde20k(img_path,
output_image_name="./Images/segmented.jpg")
```

Semantic-segmentation.ipynb

```
import numpy as np
import os
import cv2
import tensorflow as tf
import matplotlib.pyplot as plt
from skimage import io
from glob import glob
from tqdm import tqdm

train_path='cityscapes_data/train'
val_path='cityscapes_data/val'
train_images=[]
train_masks=[]
val_images=[]
val_masks=[]

def load_images(path):
    temp_img,temp_mask=[],[]
    images=glob(os.path.join(path,'*.jpg'))
    for i in tqdm(images):
        i=cv2.imread(i)
        i=cv2.normalize(i,None,0,1,cv2.NORM_MINMAX,cv2.CV_32F)
        img=i[:,:,:256]
        msk=i[:, :,256:]
        temp_img.append(img)
        temp_mask.append(msk)
    return temp_img,temp_mask

train_images,train_masks=load_images(train_path)
val_images,val_masks=load_images(val_path)

def relu(x,threshold=0.1):
    return tf.maximum(x,x*threshold)

def conv_layer(x,n_filters,k_size,stride,padding='SAME'):
    x=tf.layers.conv2d(x,filters=n_filters,kernel_size=k_size,strides=stride,padding=padding)
    x=tf.nn.relu(x)
```

```

    return x

def max_pool(x, pool_size):
    x = tf.layers.max_pooling2d(x, pool_size=pool_size)
    return x

def conv_transpose(x, n_filters, k_size, stride, padding='SAME'):
    x = tf.layers.conv2d_transpose(x, filters=n_filters, kernel_size=k_size, strides=stride, padding=padding)
    x = tf.nn.relu(x)
    return x

#Placeholders
image = tf.placeholder(tf.float32, [None, 256, 256, 3], name='Input_image')
mask = tf.placeholder(tf.float32, [None, 256, 256, 3], name='Image_mask')

#####Beta Network

#Branch-0
layer_1 = conv_layer(image, n_filters=64, k_size=4, stride=1)
mp_1 = tf.layers.max_pooling2d(layer_1, pool_size=2, strides=2)

layer_2 = conv_layer(mp_1, n_filters=128, k_size=4, stride=1)
mp_2 = tf.layers.max_pooling2d(layer_2, pool_size=2, strides=2)

layer_3 = conv_layer(mp_2, n_filters=256, k_size=4, stride=1)
mp_3 = tf.layers.max_pooling2d(layer_3, pool_size=2, strides=2)

layer_4 = conv_layer(mp_3, n_filters=512, k_size=4, stride=1)
mp_4 = tf.layers.max_pooling2d(layer_4, pool_size=2, strides=2)

layer_5 = conv_layer(mp_4, n_filters=1024, k_size=4, stride=1)
mp_5 = tf.layers.max_pooling2d(layer_5, pool_size=2, strides=2)

#Branch_1
layer_b1 = conv_layer(image, n_filters=128, k_size=4, stride=1)
mp_b1 = tf.layers.max_pooling2d(layer_b1, pool_size=2, strides=2)

beta_1 = tf.keras.layers.add([layer_2, mp_b1])

layer_b2 = conv_layer(beta_1, n_filters=256, k_size=4, stride=1)
mp_b2 = tf.layers.max_pooling2d(layer_b2, pool_size=2, strides=2)

beta_2 = tf.keras.layers.add([layer_3, mp_b2])

layer_b3 = conv_layer(beta_2, n_filters=512, k_size=4, stride=1)

```

```

mp_b3=tf.layers.max_pooling2d(layer_b3,pool_size=2,strides=2)

beta_3=tf.keras.layers.add([mp_b3,layer_4])

layer_b4=conv_layer(beta_3,n_filters=1024,k_size=4,stride=1)
mp_b4=tf.layers.max_pooling2d(layer_b4,pool_size=2,strides=2)

beta_4=tf.keras.layers.add([mp_b4,layer_5])

beta_0=layer_1
#####

#Downsample
#64
x_layer_1=conv_layer(image,n_filters=64,k_size=5,stride=1)
x_layer_1=conv_layer(x_layer_1,n_filters=64,k_size=4,stride=1)
x_layer_1=conv_layer(x_layer_1,n_filters=64,k_size=4,stride=2)
x_batch_1=tf.layers.batch_normalization(x_layer_1)#128x128x64

#128
x_layer_2=conv_layer(x_batch_1,n_filters=128,k_size=5,stride=1)
x_layer_2=conv_layer(x_layer_2,n_filters=128,k_size=4,stride=1)
x_layer_2=conv_layer(x_layer_2,n_filters=128,k_size=4,stride=2)
x_batch_2=tf.layers.batch_normalization(x_layer_2)#64x64x128

#256
x_layer_3=conv_layer(x_batch_2,n_filters=256,k_size=5,stride=1)
x_layer_3=conv_layer(x_layer_3,n_filters=256,k_size=4,stride=1)
x_layer_3=conv_layer(x_layer_3,n_filters=256,k_size=4,stride=2)
x_batch_3=tf.layers.batch_normalization(x_layer_3)#32x32x256

#512
x_layer_4=conv_layer(x_batch_3,n_filters=512,k_size=5,stride=1)
x_layer_4=conv_layer(x_layer_4,n_filters=512,k_size=4,stride=1)
x_layer_4=conv_layer(x_layer_4,n_filters=512,k_size=4,stride=2)
x_batch_4=tf.layers.batch_normalization(x_layer_4)#16x16x512

#1024
x_layer_5=conv_layer(x_batch_4,n_filters=1024,k_size=4,stride=1)
x_layer_5=conv_layer(x_layer_5,n_filters=1024,k_size=4,stride=8)
x_batch_5=tf.layers.batch_normalization(x_layer_5)#8x8x1024

#Upsample
#1024
y_layer_1=conv_transpose(x_batch_5,n_filters=1024,k_size=4,stride=8)

```

```

y_layer_1=tf.keras.layers.add([y_layer_1,beta_4])
y_layer_1=conv_layer(y_layer_1,n_filters=1024,k_size=4,stride=1)
y_batch_1=tf.layers.batch_normalization(y_layer_1)

#512
y_layer_2=conv_transpose(y_batch_1,n_filters=512,k_size=5,stride=2)
y_layer_2=tf.keras.layers.add([y_layer_2,beta_3])
y_layer_2=conv_layer(y_layer_2,n_filters=512,k_size=4,stride=1)
y_layer_2=conv_layer(y_layer_2,n_filters=512,k_size=4,stride=1)
y_batch_2=tf.layers.batch_normalization(y_layer_2)

#256
y_layer_3=conv_transpose(y_batch_2,n_filters=256,k_size=5,stride=2)
y_layer_3=tf.keras.layers.add([y_layer_3,beta_2])
y_layer_3=conv_layer(y_layer_3,n_filters=256,k_size=4,stride=1)
y_layer_3=conv_layer(y_layer_3,n_filters=256,k_size=4,stride=1)
y_batch_3=tf.layers.batch_normalization(y_layer_3)

#128
y_layer_4=conv_transpose(y_batch_3,n_filters=128,k_size=3,stride=2)
y_layer_4=tf.keras.layers.add([y_layer_4,beta_1])
y_layer_4=conv_layer(y_layer_4,n_filters=128,k_size=2,stride=1)
y_layer_4=conv_layer(y_layer_4,n_filters=128,k_size=2,stride=1)
y_batch_4=tf.layers.batch_normalization(y_layer_4)

#64
y_layer_5=conv_transpose(y_batch_4,n_filters=64,k_size=2,stride=2)
y_layer_5=tf.keras.layers.add([y_layer_5,beta_0])
y_layer_5=conv_layer(y_layer_5,n_filters=64,k_size=1,stride=1)
y_layer_5=conv_layer(y_layer_5,n_filters=64,k_size=1,stride=1)
y_batch_5=tf.layers.batch_normalization(y_layer_5)

#Output
out=tf.layers.conv2d(y_batch_5,activation=None,filters=3,kernel_size=1,strides=1,padding='S
AME')
loss=tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(labels=mask,logits=out))
train_op=tf.train.AdamOptimizer(learning_rate=0.0001).minimize(loss)
num_epochs=100
batch_size=25
train_batches=len(train_images)//batch_size
val_batches=len(val_images)//batch_size
train_loss,val_loss=[],[]
saver=tf.train.Saver()
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for epoch in range(num_epochs):
        print('=====')

```

```

print('Epoch: ',(epoch+1))
print('In Training..')
for batch in tqdm(range(train_batches)):
    train_img_batch=train_images[batch*batch_size:(batch+1)*batch_size]
    train_msk_batch=train_masks[batch*batch_size:(batch+1)*batch_size]
    #reshape images and masks
    train_img_batch=np.reshape(train_img_batch,(len(train_img_batch),256,256,3))
    train_msk_batch=np.reshape(train_msk_batch,(len(train_msk_batch),256,256,3))
    t_loss,_=sess.run([loss,train_op],feed_dict={
        image:train_img_batch,mask:train_msk_batch})

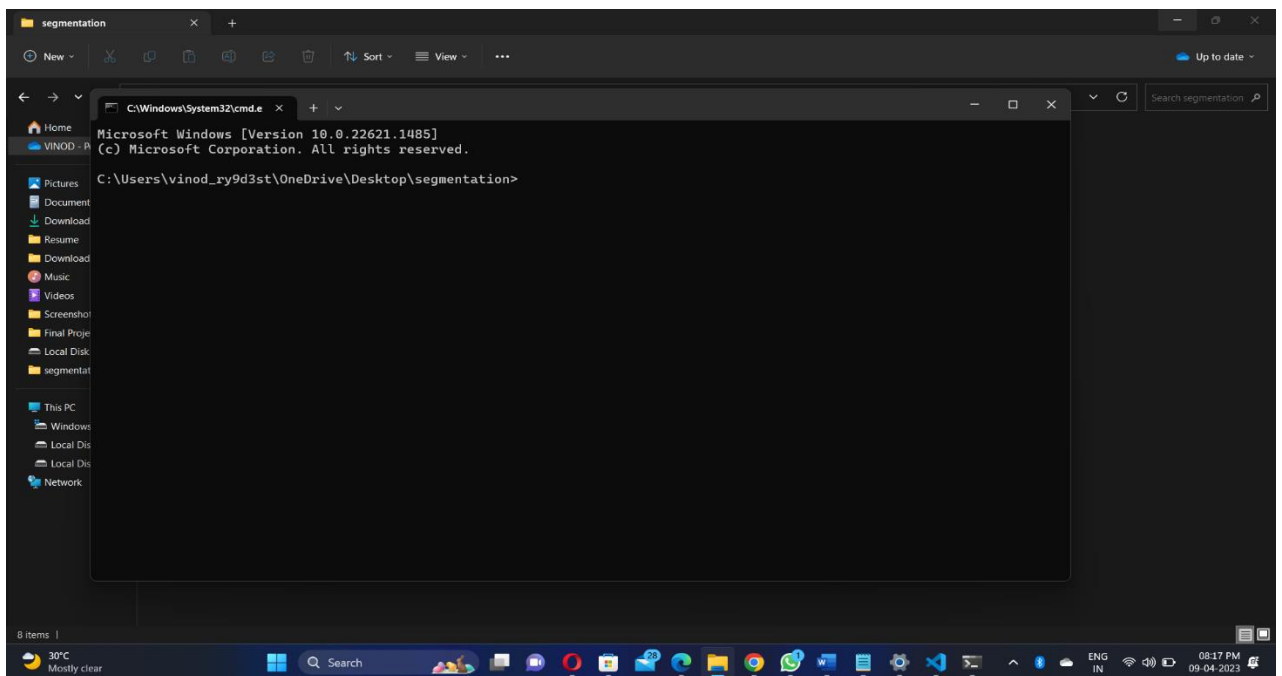
print('In Validation..')
for v_batch in tqdm(range(val_batches)):
    val_img_batch=val_images[v_batch*batch_size:(v_batch+1)*batch_size]
    val_msk_batch=val_masks[v_batch*batch_size:(v_batch+1)*batch_size]

    #Reshape batches
    val_img_batch=np.reshape(val_img_batch,(len(val_img_batch),256,256,3))
    val_msk_batch=np.reshape(val_msk_batch,(len(val_msk_batch),256,256,3))
    v_loss,_=sess.run([loss,train_op],feed_dict={image:val_img_batch,mask:val_msk_batch
h))
    train_loss.append(t_loss)
    val_loss.append(v_loss)

print('Train Loss: ',t_loss)
print('Val Loss: ',v_loss)
saver.save(sess, "model.ckpt")
saver=tf.train.Saver()
img_=train_images[5:10]
msk_=train_masks[5:10]
preds=[]
w,h=256,256
col,row=2,2
#ckpt_path='../input/semantic-segmentation-code/model.ckpt'
model_path='model.ckpt'
with tf.Session() as sess_1:
    saver.restore(sess_1,model_path)
    print('Model Restored!')
    seg=sess_1.run(out,feed_dict={image:img_})
    seg=sess_1.run(tf.nn.sigmoid(seg))
    fig=plt.figure(figsize=(8,8))
    for i in range(1,row*col+1):
        fig.add_subplot(row,col,i)
        plt.imshow(seg[i-1])
    plt.show()

```

SCREEN SHOTS: OUTPUT SCREEN SHOTS WITH DESCRIPTION.




```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.22621.1485]
(c) Microsoft Corporation. All rights reserved.

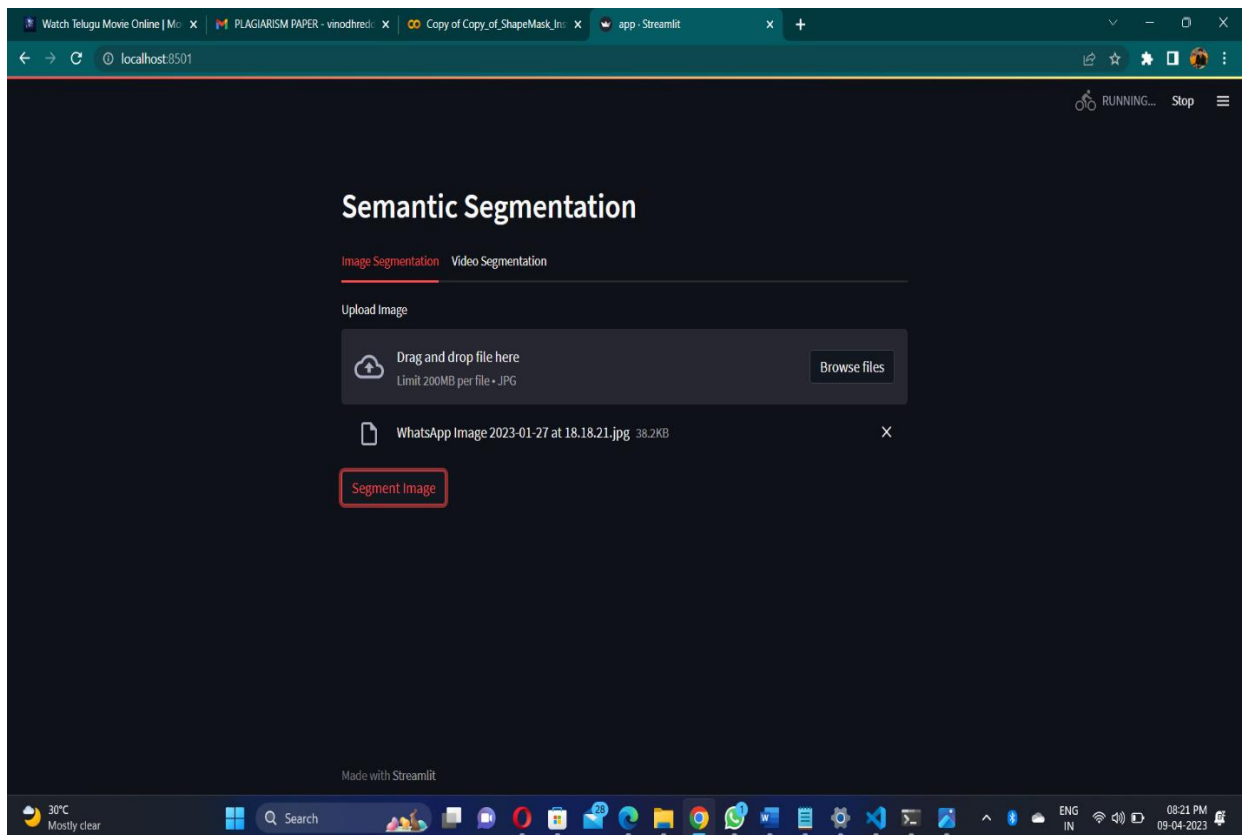
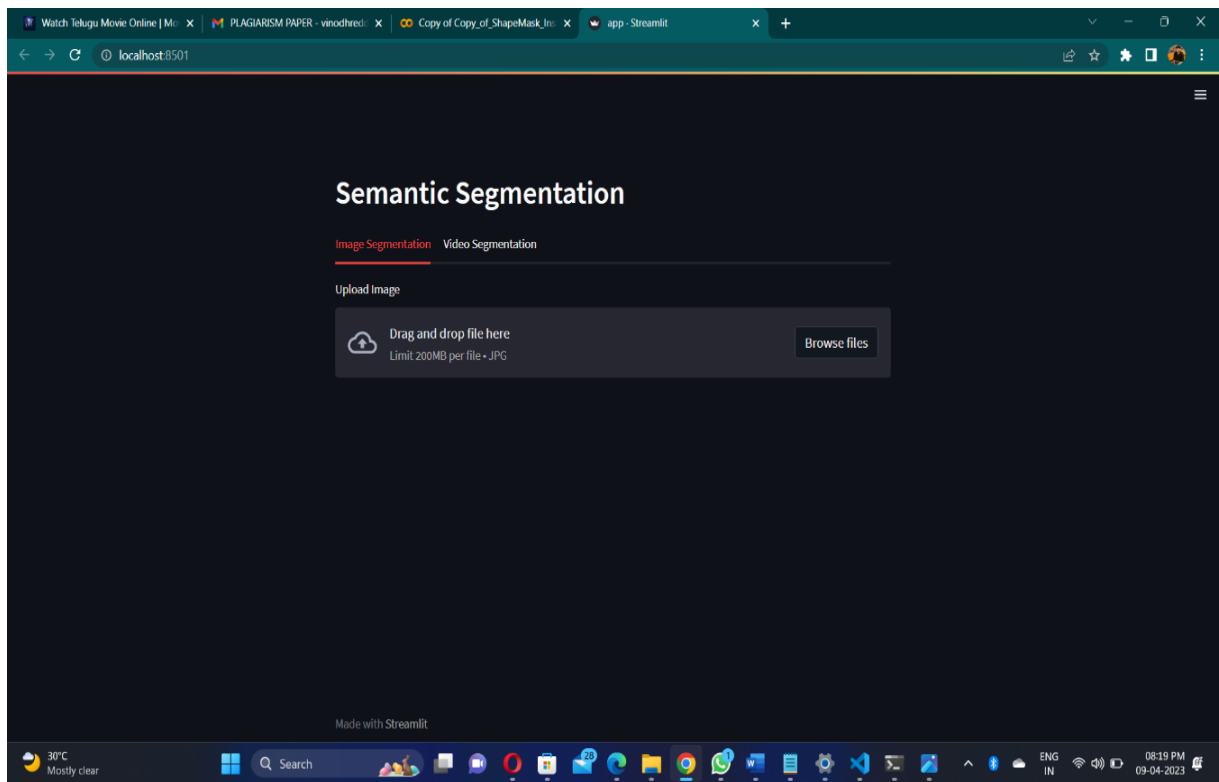
C:\Users\vinod_ry9d3st\OneDrive\Desktop\segmentation>streamlit run app.py
```

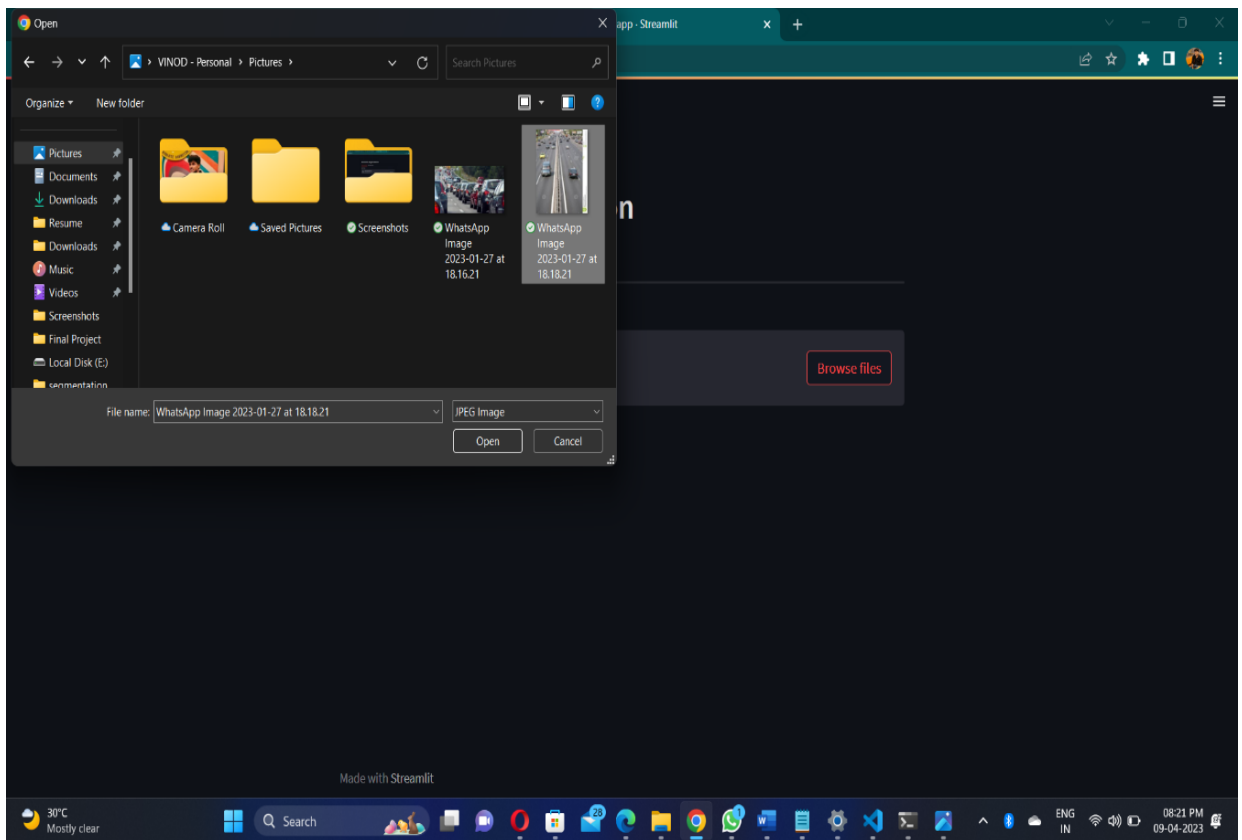
```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.22621.1485]
(c) Microsoft Corporation. All rights reserved.

C:\Users\vinod_ry9d3st\OneDrive\Desktop\segmentation>streamlit run app.py

You can now view your Streamlit app in your browser.

Local URL: http://localhost:8501
Network URL: http://192.168.1.5:8501
```





INPUT IMAGE:



INPUT IMAGE:



THANK YOU