

ROAD OBSTACLE DETECTION USING RESIDUAL FUSION NETWORK

A PROJECT REPORT

Submitted by

AKASH KIRTHIK G - 2018103006

PAARGAV SHANKER SU - 2018103048

RISHEEK RAKSHIT S K - 2018103580

*in partial fulfillment for the award of the degree
of*

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING



COLLEGE OF ENGINEERING GUINDY

ANNA UNIVERSITY : CHENNAI 600 025

JUNE 2022

ANNA UNIVERSITY : CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report **ROAD OBSTACLE DETECTION USING RESIDUAL FUSION NETWORK** is the bonafide work of **AKASH KIRTHIK G, PAARGAV SHANKER SU, RISHEEK RAKSHIT S K**, who carried out the project work under my supervision for the fulfillment of the requirements for the award of the degree of Bachelor of Engineering in Computer Science and Engineering. Certified further that to the best of my knowledge, the work reported herein does not form part of any other thesis or dissertation on the basis of which a degree or an award was conferred on an earlier occasion on these or any other candidates.

SIGNATURE

Dr. S. Valli

HEAD OF THE DEPARTMENT

Computer Science and Engineering
Anna University, Guindy, Chennai,
Tamil Nadu 600025

SIGNATURE

Dr. T. V. Gopal

SUPERVISOR

Professor
Computer Science and Engineering
Anna University, Guindy, Chennai,
Tamil Nadu 600025

ACKNOWLEDGEMENT

We express our deep gratitude to our guide, **Dr. T. V. Gopal**, Professor, Department of Computer Science and Engineering for guiding us through every phase of the project. We appreciate his thoroughness, tolerance, and ability to share his knowledge with us. We would also like to thank him for his kind support and for providing the necessary facilities to carry out the work.

We are incredibly grateful to **Dr. S. Valli**, Professor & Head of the Department, Department of Computer Science and Engineering, Anna University, Chennai – 25, for extending the facilities of the Department towards our project and for her unstinting support.

We express our thanks to the panel of reviewers **Dr. S. Chitrakala**, Professor, Department of Computer Science and Engineering, **Dr. V. Vetrisevi**, Professor, Department of Computer Science and Engineering and **Dr. G.S. Mahalakshmi**, Associate Professor, Department of Computer Science and Engineering for their valuable suggestions and critical reviews throughout the course of our project.

We express our thanks to all other teaching and nonteaching staff who helped us in one way or another for the successful completion of the project. We would also like to thank our parents, family, and friends for their indirect contribution to successfully completing this project.

AKASH
KIRTHIK G

PAARGAV
SHANKER SU

RISHEEK
RAKSHIT S K

TABLE OF CONTENTS

	ABSTRACT - ENGLISH	vi
	ABSTRACT - TAMIL	vii
	LIST OF TABLES	viii
	LIST OF FIGURES	ix
	LIST OF ABBREVIATIONS	xi
1.	INTRODUCTION	
	1.1 Introduction	1
	1.2 Problem Statement	2
	1.2 Overall Objectives	2
2.	RELATED WORKS	3
3.	SYSTEM DESIGN	
	3.1 System Architecture	6
	3.2 Model Architecture	7
	3.3 Detailed Module Design	8
	3.3.1. Data Preprocessing	8
	3.3.2. RGB-D Fusion	9
	3.3.3. Obstacle Segmentation	11
	3.3.4. Video Segmentation	12
4.	IMPLEMENTATION AND RESULTS	
	4.1 Dataset Description	14
	4.1.1 Cityscapes	14

4.1.2 Lost and Found	16
4.2 Implementation Details	18
4.2.1 Data Preprocessing	18
4.2.1.1 Disparity Map Generation	18
4.2.1.2 Data Preprocessing	19
4.2.1.3 Generating Train Labels	21
4.2.2 Model Definition	22
4.2.2.1 RGB Layer	22
4.2.2.2 Depth Layer	23
4.2.2.3 Attention Feature Complementary	24
4.2.2.4 Spatial Pyramid Pooling	25
4.2.2.5 Upsampling	25
4.2.3 Training and Validation	27
4.2.3.1 Loading Images	27
4.2.3.2 Training	27
4.2.3.3 Hyperparameters for Training	30
4.2.3.4 Evaluation	31
4.2.3.5 Video Segmentation	32
4.3 Results and Test Cases	35
4.4 Performance Metrics	38
4.5 Performance Graphs	40
4.6 Comparative Analysis	42

5.	CONCLUSION AND FUTURE EXPANSION	
	5.1 Conclusion	43
	5.2 Future Work	43
	REFERENCES	44

ABSTRACT - ENGLISH

Semantic segmentation has made striking progress due to the success of deep convolutional neural networks. Considering the demands of autonomous driving, real-time semantic segmentation has become a research hotspot these years. However, few real-time RGB-D fusion semantic segmentation studies are carried out despite readily accessible depth information nowadays.

We propose a real-time fusion semantic segmentation network termed RFNet that effectively exploits complementary cross-modal information. with the advent of more powerful hardware and recent advancements in Neural Networks, it is now much easier to implement such designs. Building on an efficient network architecture, RFNet is capable of running swiftly, which satisfies autonomous vehicles applications. For autonomous driving, identifying a wide range of obstacles at reasonable speed with good accuracy is required. Multi-dataset training is leveraged to incorporate unexpected small obstacle detection, enriching the recognizable classes required to face unforeseen hazards in the real world.

சுருக்கம் - தமிழ்

ஆழமான கன்வல்யூஷனல் நியூரல் நெட்வொர்க்குகளின் வெற்றியின் காரணமாக சொற்பொருள் பிரிவு குறிப்பிடத்தக்க முன்னேற்றத்தை அடைந்துள்ளது. தன்னாட்சி வாகனம் ஓட்டுவதற்கான கோரிக்கைகளைக் கருத்தில் கொண்டு, நிகழ்நேர சொற்பொருள் பிரிவு இந்த ஆண்டுகளில் ஒரு ஆராய்ச்சி மையமாக மாறியுள்ளது. இருப்பினும், இப்போதெல்லாம் உடனடியாக அணுகக்கூடிய ஆழமான தகவல் இருந்தபோதிலும், சில நிகழ்நேர *RGB-D* இணைவு சொற்பொருள் பிரிவு ஆய்வுகள் மேற்கொள்ளப்படுகின்றன.

RFNet எனப்படும் நிகழ்நேர இணைவு சொற்பொருள் பிரிவு வலையமைப்பை நாங்கள் முன்மொழிகிறோம், இது நிரப்பு குறுக்கு மாதிரி தகவலை திறம்பட பயன்படுத்துகிறது. மிகவும் சக்திவாய்ந்த வன்பொருள் மற்றும் நரம்பியல் நெட்வொர்க்கில் சமீபத்திய முன்னேற்றங்களின் வருகையுடன், அத்தகைய வடிவமைப்புகளை செயல்படுத்துவது இப்போது மிகவும் எளிதானது. திறமையான நெட்வொர்க் கட்டமைப்பை உருவாக்கி, *RFNet* விரைவாக இயங்கும் திறன் கொண்டது, இது தன்னாட்சி வாகனங்களின் பயன்பாடுகளை திருப்திப்படுத்துகிறது. தன்னாட்சி வாகனம் ஓட்டுவதற்கு, நல்ல துல்லியத்துடன் நியாயமான வேகத்தில் பரந்த அளவிலான தடைகளை அடையாளம் காண வேண்டும். மல்டி-டேட்டாசெட் பயிற்சியானது, எதிர்பாராத சிறிய தடைகளைக் கண்டறிவதை இணைத்து, நிஜ உலகில் எதிர்பாராத ஆபத்துக்களை எதிர்கொள்ளத் தேவையான அடையாளம் காணக்கூடிய வகுப்புகளை வளப்படுத்துகிறது.

LIST OF TABLES

Table 4.1	Hyperparameters for training	30
Table 4.2	Video frames and its corresponding prediction	32
Table 4.3	Possible test cases with description	35
Table 4.4	Performance metrics	41
Table 4.5	Comparison of semantic segmentation methods on the validation set	42

LIST OF FIGURES

Figure 3.1	System Architecture	6
Figure 3.2	Model Architecture	7
Figure 3.3	Data Preprocessing Module Diagram	8
Figure 3.4	RGB-D Fusion Module Diagram	10
Figure 3.5	Obstacle Segmentation Module Diagram	12
Figure 3.6	Video Segmentation Module Diagram	13
Figure 4.1	Different cities in Cityscapes dataset	14
Figure 4.2	A glimpse of Cityscapes dataset images	15
Figure 4.3	Cityscapes Dataset Sample Images	15
Figure 4.4	Different cities in Lost and Found dataset	16
Figure 4.5	A glimpse of Lost and Found dataset	17
Figure 4.6	Lost and Found Dataset Sample Images	17
Figure 4.7	Generating Disparity Maps	18
Figure 4.8	Cropping Black Area	19
Figure 4.9	Performing Random Horizontal Flip	19
Figure 4.10	Performing Random Scale Crop	20
Figure 4.11	Performing Gaussian Blur	21
Figure 4.12	Generating Train Labels	21
Figure 4.13	Defining RGB Layer of RFNet	22
Figure 4.14	Extracted Features from RGB Layer 1	22
Figure 4.15	Defining Depth Layer of RFNet	23
Figure 4.16	Extracted Features from Depth Layer 1	23
Figure 4.17	Defining Attention Feature Complementary Module	24
Figure 4.18	Extracted Features from AFC 1	24
Figure 4.19	Defining Spatial Pyramid Pooling Module	25
Figure 4.20	Defining Upsampling	26
Figure 4.21	Extracted Features from Upsampling 3	26

Figure 4.22	Loading images and initializing epochs	27
Figure 4.23	0th and 1st epoch training with loss	27
Figure 4.24	Accuracy and IoU of each class	28
Figure 4.25	First validation with performance metrics	28
Figure 4.26	Last two epoch training with loss	29
Figure 4.27	Accuracy and IoU of each class	29
Figure 4.28	Last validation with performance metrics	30
Figure 4.29	Loading images and performing evaluation/testing	31
Figure 4.30	Accuracy and IoU of each class	31
Figure 4.31	Performance metrics for evaluation/testing	32
Figure 4.32	Legend for obstacles with corresponding color	35
Figure 4.33	Validation Accuracy Graph	40
Figure 4.34	Class Accuracy Graph	40
Figure 4.35	Training Loss Graph	40
Figure 4.36	Validation Loss Graph	40
Figure 4.37	Mean Intersection Over Union Graph	40
Figure 4.38	Frequency Weighted Mean Intersection Over Union Graph	40
Figure 4.39	Precision Graph	41
Figure 4.40	Recall Graph	41
Figure 4.41	F1 Score Graph	41
Figure 4.42	Different models comparison of mIoU value	42

LIST OF ABBREVIATIONS

AFC - Attention Feature Complementary
CNN - Convolutional Neural Network
FWIoU - Frequency Weighted Intersection over Union
GFU - Gated Fusion Unit
GPU - Graphics Processing Unit
LiDAR - Light Detection and Ranging
mIoU - mean Intersection over Union
RELU - Rectified Linear Unit
ResNet - Residual neural Network
RFNET - Residual Fusion Network
RGB - Red Green Blue
RGBA - Red Green Blue Alpha
RGB-D - Red Green Blue Depth
RU - Residual Unit
SE block - Squeeze and Excitation block
SGM - Semi Global Matching
SPP - Spatial Pyramid Pooling

CHAPTER 1

INTRODUCTION

1.1 Introduction

Environment perception is important for intelligent robots and systems in object classification, autonomous driving, and location. Deep Convolutional Neural Networks (CNNs)-based semantic segmentation methods have made significant progress in this field in recent years.

In the year 2020, there were 449,002 accidents on Indian roads. About one fourth of these result in at least one casualty. A careless driver not only poses a threat to themselves, but also to other vehicles and pedestrians. There is a requirement for a reasonably accurate model capable of detecting road obstructions. There are multiple ways to detect obstructions like sound waves, LiDAR, etc. Our goal is to develop an image-based road obstacle detection system. Although Obstacle detection has been a heavily researched topic for decades, spanning a variety of application areas, only little work has focused on the detection of small, generic and unexpected obstacles in driving scenarios and there aren't many systems in place that would warn the driver of upcoming obstacles.

Unexpected road hazards such as debris, bricks, stones, and cargos become the most harmful and hardest to identify in autonomous driving images. Road obstacles cause safety hazards that result in many vehicular crashes. These obstacles are generally small in size but not fixed in shape and type, making detecting them a challenging subject among the robotics and computer vision community. A system that would be able to identify such road obstacles in real time would be hugely beneficial in many ways. We mainly consider camera-based methods for the detection and localization of such generic obstacles in 3D space, using stereo camera setups with small baselines (i.e. less than 25 cm). Due

to recent developments in deep convolutional neural networks, semantic segmentation has advanced dramatically, both in terms of efficiency and accuracy. The challenges of autonomous driving include identification of all types of obstacles, accuracy and fast processing speed. Given these challenges, real-time semantic segmentation has emerged to be an in-demand research topic in recent years.

1.2 Problem Statement

Due to ever increasing numbers of road vehicles, the number of accidents also have increased dramatically in recent years. Accidents occur mostly due to driver negligence. A suitable warning system would be highly beneficial to warn the driver about obstacles present on the road.

In autonomous driving photos, unexpected road dangers such as debris, bricks, stones, and cargos become the most dangerous and difficult to identify. Road hazards pose a safety risk, resulting in a high number of vehicle collisions. These impediments are often modest in size but vary in shape and type, making detection a difficult task for the robotics and computer vision communities.

1.3 Overall Objectives

1. To develop a semantic segmentation-based method incorporating pixel-wise unexpected obstacle detection.
2. To incorporate a multi-dataset training strategy to detect a wide variety of road obstacles, including smaller ones.
3. To build an application which can detect road obstacles in videos.

CHAPTER 2

RELATED WORKS

C. Couprie, C. Farabet, et al. [6] used depth information to recognize objects of classes having similar depth appearance and location is improved when using depth information. It uses separate CNN for RGB and depth images. This suggests that recognition architectures for vision (and for other modalities such as audio and natural language) should have multiple trainable stages stacked on top of each other, one for each level in the feature hierarchy. The paper proposes the use of multiscale convolutional network to learn features directly from the images and the depth information. In multi-scale feature extraction, pixels are assembled into edglets, edglets into motifs, motifs into parts, parts into objects, and objects into scenes. Though it classifies objects correctly, it performs poorly when it comes to outdoor conditions and hence it cannot be applied to object detection in roads. The indoor dataset contains scenes of offices, stores, rooms of houses containing many occluded objects unevenly lightened, in contrast to outdoors which are typically bright.

S. Ramos, S. Gehrig, et al. [7] uses deep learning methods to detect small, generic obstacles (pixel-wise segmentation), training the model on 2-D images and then using a probabilistic Bayesian framework to fuse the current model and existing models which were trained on stereo images. Though it performs well in outdoor conditions, previously unseen objects cannot be detected which is crucial for autonomous cars. It demonstrates how the ability of CNNs to learn context and generalize information from training data can overcome one of their main open problems: handling outliers and the “open world”. It presents a probabilistic fusion approach that combines our learning-based detection method with the currently best performing stereo-based system for this task. Under real world conditions, an obstacle would not have a well-defined shape or size. This

introduces a new set of challenges that would require identification of previously unseen objects. Even though the model performs well in outdoor conditions, previously unseen objects cannot be detected which is crucial for autonomous cars.

W. Wang and U. Neumann [5] uses depth-aware convolution and depth-aware average pooling. Depth-aware convolution augments the standard convolution with a depth similarity term. We force pixels with similar depths with the center of the kernel to have more contribution to the output than others. This simple depth similarity term efficiently incorporates geometry in a convolution kernel and helps build a depth-aware receptive field, where convolution is not constrained to the fixed grid geometric structure. The second introduced operator is depth-aware average pooling. Similarly, when a filter is applied on a local region of the feature map, the pairwise relations in depth between neighboring pixels are considered in computing mean of the local region. Visual features are able to propagate along with the geometric structure given in depth images. Such geometry-aware operation enables the localization of object boundaries with depth images. However, to obtain optimal results, dense depth maps are required for the CNN to fully exploit the information provided.

L. Deng, M. Yang, et al. [4] proposed a novel bottom-up interactive fusion structure to model the interdependencies between the encoders. The structure introduces an interaction stream to interconnect the encoders. The interaction stream not only progressively aggregates modality-specific features from the encoders but also computes complementary features for them. To instantiate this structure, the paper proposes a residual fusion block (RFB) to formulate the interdependencies of the encoders. The RFB consists of two residual units and one fusion unit with gate mechanism. It learns complementary features for the modality-specific encoders and extracts modality-specific features as well as

cross-modal features. Based on the RFB, the paper presents the deep multimodal networks for RGBD semantic segmentation called RFBNets. The RFB consists of two modality-specific residual units (RUs) and one gated fusion unit (GFU). The GFU adaptively aggregates features from the RUs and generates complementary features for the RUs. The RFB formulates the complementary feature learning as residual learning, and it can extract modality-specific and cross-modal features. With the RFBs, the modality-specific encoders can interact with each other. Though it performs well in terms of how accurate it can predict the obstacles, it is a bulkier module as it uses residual fusion blocks. Real-time processing of inputs is difficult due to its bulkiness. This would require expensive Graphics Processing Units (GPU) to perform the bulky computations.

CHAPTER 3

SYSTEM DESIGN

3.1 System Architecture

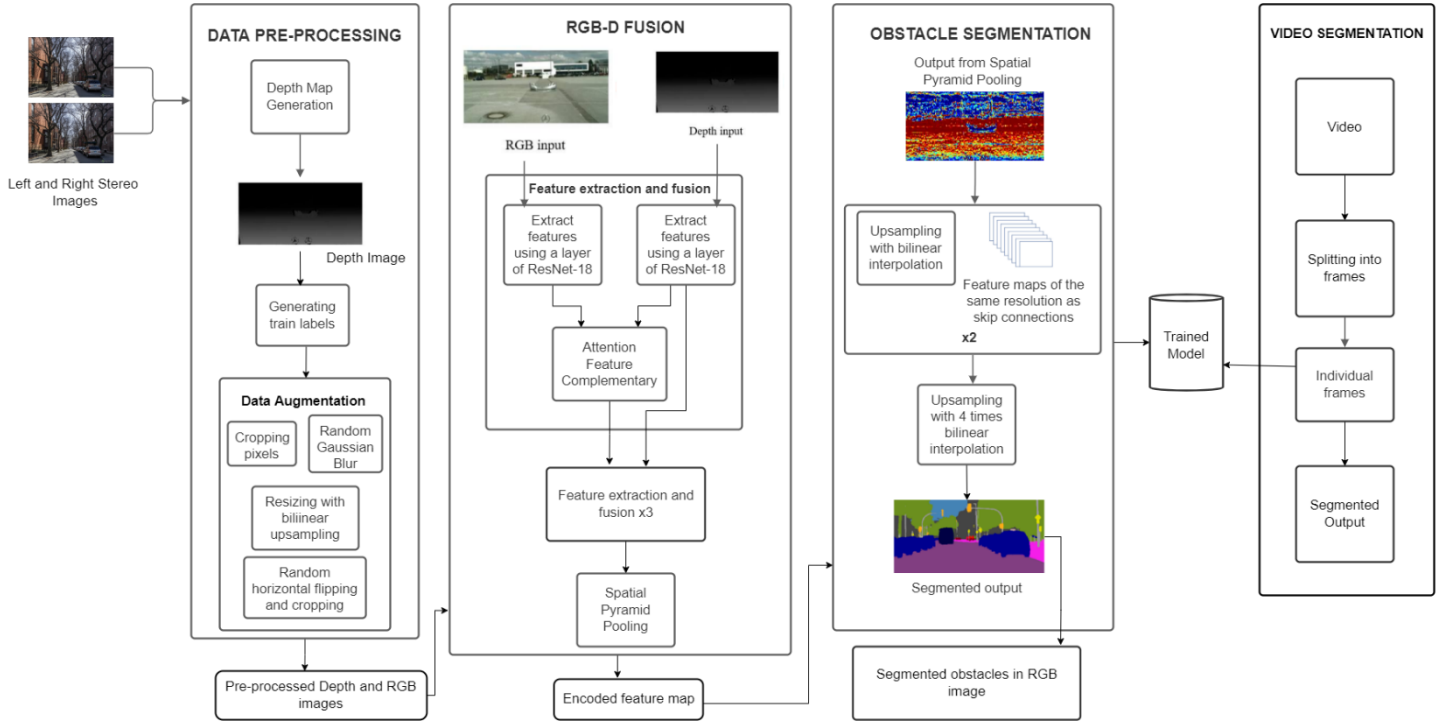


Figure 3.1 - System Architecture

The input left and right stereo images are used to generate disparity maps which are then preprocessed. The features are extracted using ResNet-18 and then the obstacles are segmented. Figure 3.1 depicts the whole system's workflow, in which input is passed from module to module through ResNet-18 layers.

3.2 Model Architecture

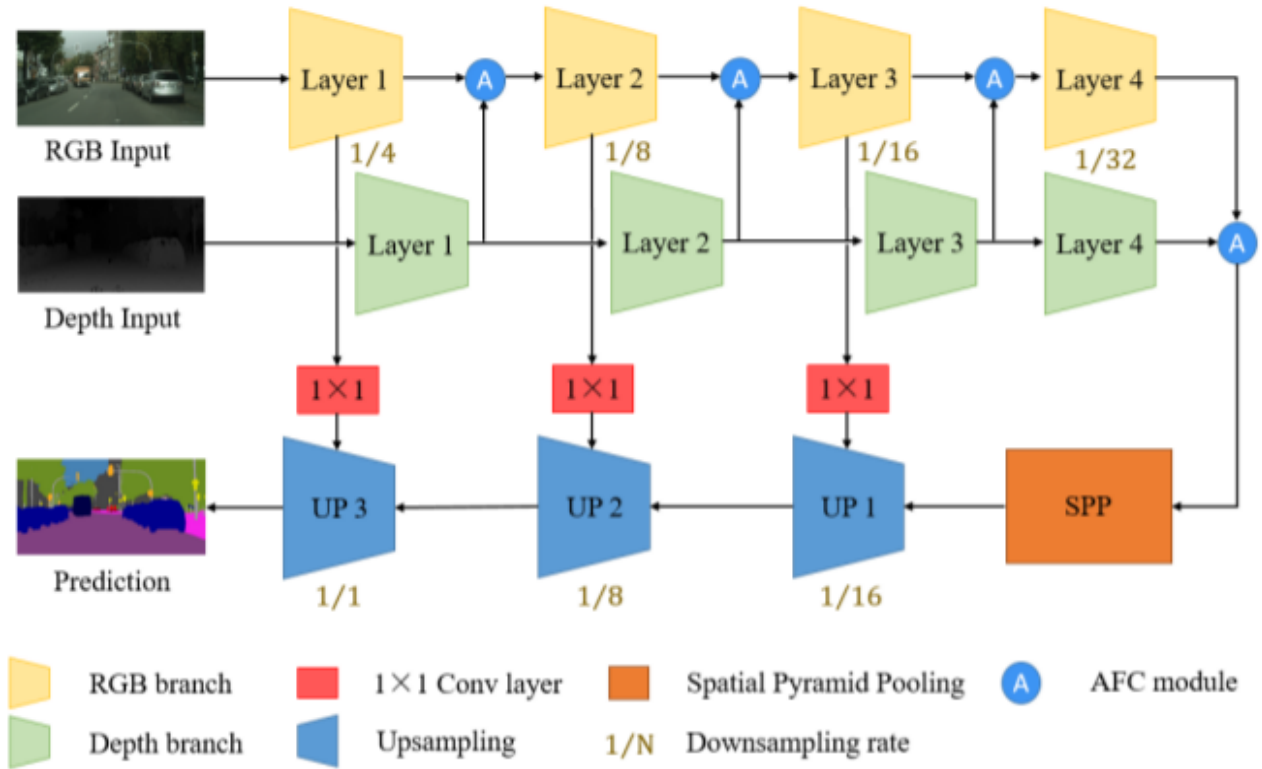


Figure 3.2 – Model Architecture

Figure 3.2 depicts the architecture of the system. The encoder part of the model consists of a RGB layer and a Depth layer, each layer consists of a ResNet-18. The layers in the figure depicts the blocks in ResNet-18 architecture. The feature maps from both these layers are fused using the Attention Feature Complementary module. The decoder segment consists of three upsampling modules to upsize the initial predictions to the size of the input image.

3.3 Detailed Module Design

3.3.1 Data Preprocessing

We generate disparity maps with the help of the Semi-Global matching algorithm. The input to this algorithm is a pair of images termed left and right images and the output is the disparity map which is required for the following module. The disparity map generated will have blacked pixels at the bottom as well as at the left. We have to crop that as part of pre-processing and we also have to resize images back to the original resolution with bilinear upsampling. The rest of the data augmentation operations consist of scaling with random factors, random horizontal flipping, and random cropping with an output resolution of 768 x 768. The overall process is shown in Figure 3.3.

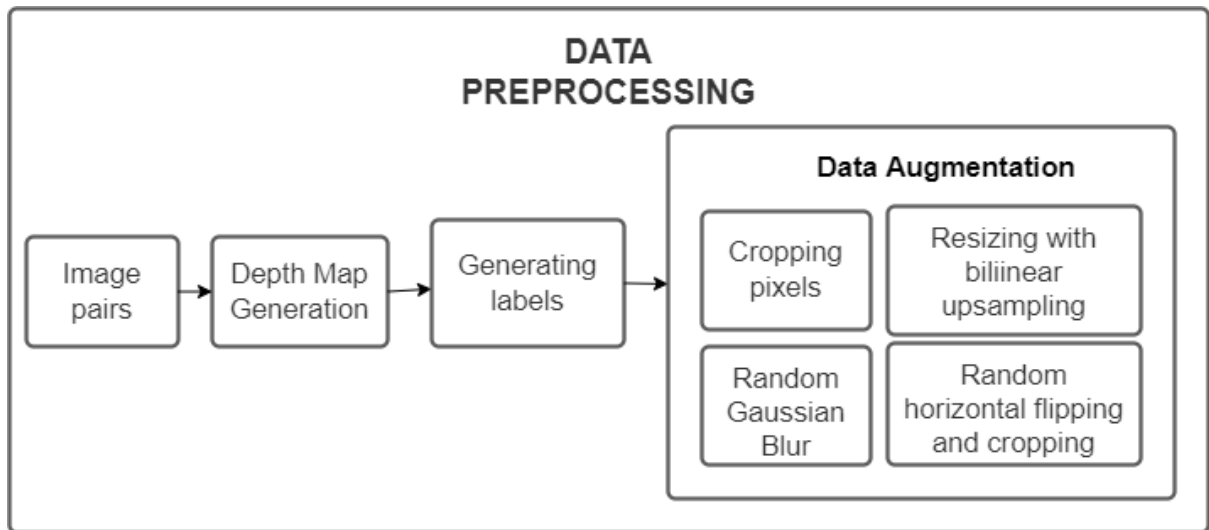


Figure 3.3 – Data Preprocessing Module Diagram

Input: Stereo Image Pairs

Output: Preprocessed image with its corresponding disparity maps

Pseudo code

Generate Disparity Maps

```
for each stereo pair:
    left,right = read(leftImage, rightImage)
    stereo = StereoBM_create()
    disparity = stereo.compute()
```

Preprocessing

```
for each depth image:
    croppedDepthImage = cropBlackArea()
    upsampling(croppedDepthImage)
for each left image, depth image:
    randomlHorizontalFlip()
    randomCrop()
for each left image:
    randomGaussianBlur()
```

Generate Train Labels

```
colorMap = [[color code for each obstacle]]
for every image:
    rgb = splitChannels()
    newLabel = [ ]
    for each pixel:
        for class in colorMap:
            if (rgb[pixel][0], rgb[pixel][1], rgb[pixel][2]) == class:
                newLabel[pixel] = class
    generateImage(newLabel)
```

3.3.2 RGB-D Fusion

The depth maps have more contour and location information, which aids RGB semantic segmentation. To effectively fuse RGB and depth information, the Attention Feature Complementary (AFC) module instructs the network to learn more complementary informative features from the RGB and Depth branches.

Each branch is made up of four ResNet-18 blocks that are connected as shown in the model architecture. With SE block as the channel attention method in the AFC module, it learns to use global information to emphasize informative channels while suppressing less useful channels, allowing the AFC module to effectively exploit informative features from both branches. The merged feature maps include detailed high-level semantic information. We use Spatial Pyramid Pooling (SPP) to average features over aligned grids with different granularities before upsampling to increase the receptive field to cover pixels of large objects while maintaining real-time speed. The SPP layer pools the features and generates fixed-length outputs, which are then fed into the fully-connected layers. The overall process is shown in Figure 3.4.

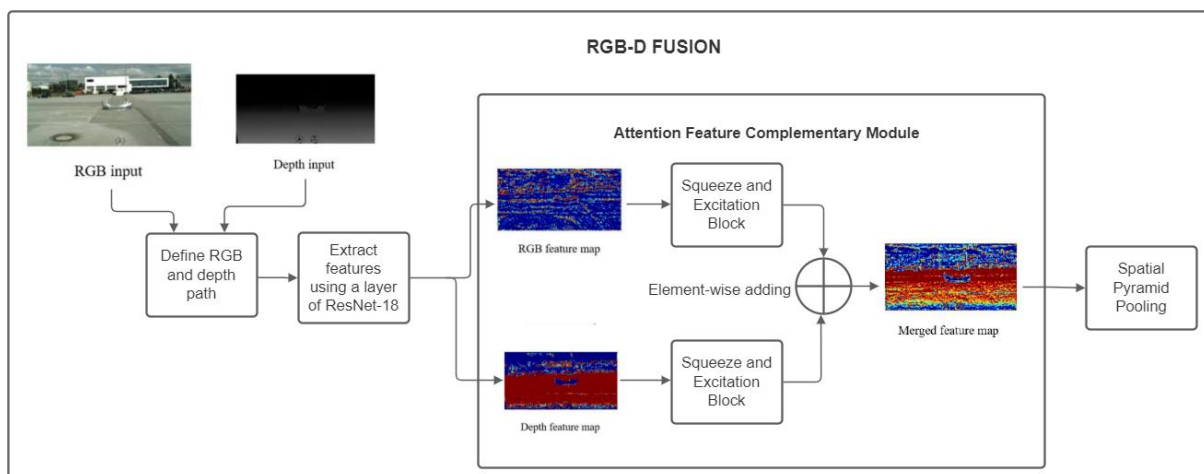


Figure 3.4 - RGB-D Fusion Module Diagram

Input: Depth map and image

Output: Encoded feature map

Pseudo code

RGB-D Fusion

```
class ResNet():
    defineResNetBlocks()
    define AFC()

rgbResnet = ResNet(weightsImagenet)
depthResnet = ResNet(weightsImagenet)
for blockNo in range(1,4):
    outputRGB = forwardPassRGB(RGBimage,blockNo)
    outputDepth = forwardPassDepth(depthMap,blockNo)
    featureMap = afc(outputRGB,outputDepth)
    RGBimage = featureMap
    depthMap = outputDepth

encodedFeatureMap = SPP(afc(outputRGB,outputDepth))
```

3.3.3 Obstacle Segmentation

The purpose of the decoder is to upsample semantically rich visual features in coarse spatial resolution to the input resolution. We adopt a simple decoder that contains three simple upsampling modules with skip connections from the encoder. In the first two upsampling modules, low-resolution feature maps from the former block are upsampled with bilinear interpolation to the same resolution as feature maps from skip connection. The third upsampling module is slightly different because we add a convolution layer and a 4-times bilinear interpolation at last to restore to the same resolution as the input. The overall process is shown in Figure 3.5.

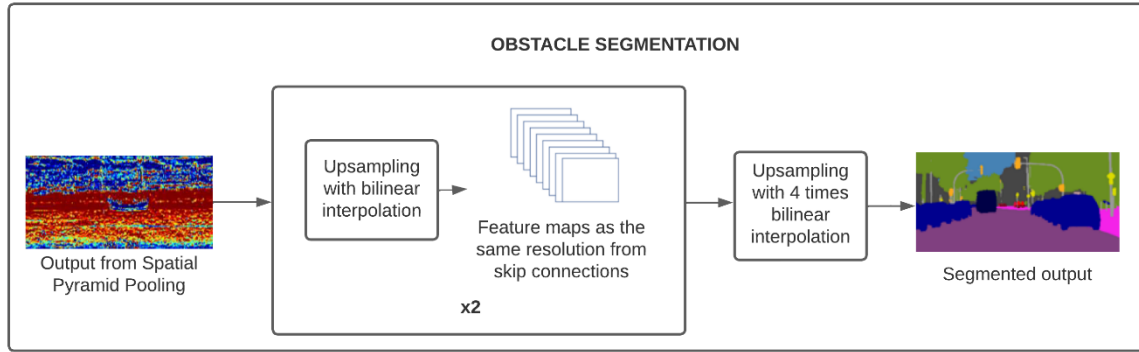


Figure 3.5 – Obstacle Segmentation Module Diagram

Input: Encoded feature map

Output: Segmented obstacles

Pseudo code

Obstacle Segmentation

```
encodedFatureMap = SPP(afc(outputRGB,outputDepth))
```

```
for block in range(3,1):
```

```
    trace = Conv1x1(skipConnection,block)
```

```
    skipDimensions = size(trace)
```

```
    val = bilinearInterpolation(featureMap,skipDimensions)
```

```
    featureMap = Merge(trace,val)
```

```
bilinearInterpolation(featureMap, scale = 4)
```

3.3.4 Video Segmentation

To make this model work on videos, we split the real time video frame-by-frame and that is fed into the exported trained model which will give the segmented output. This output can be used for warning the driver about the obstacles. The overall process is shown in Figure 3.6.

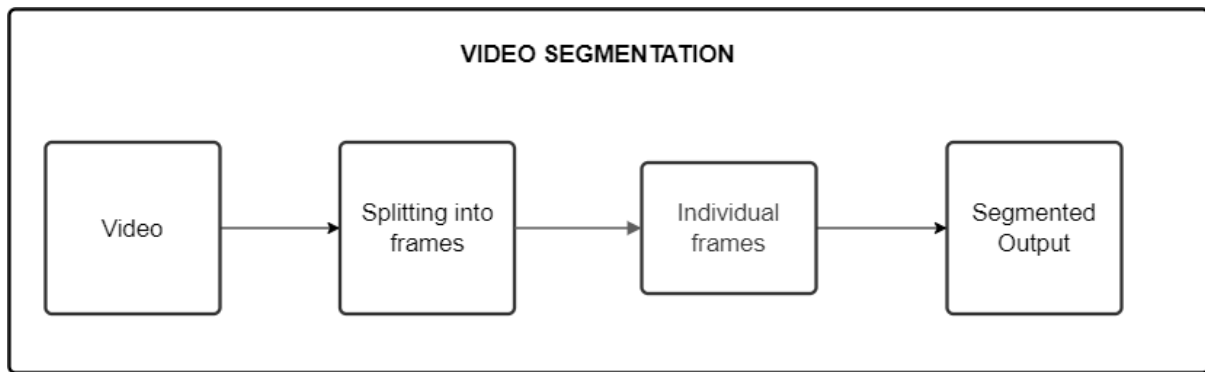


Figure 3.6 – Video Segmentation Module Diagram

Input: Video feed

Output: Segmented images

Pseudo code

Segment Obstacles

```
frames = splitIntoFrames(video)
for each frame in frames:
    prediction = model.predict(frame)
```

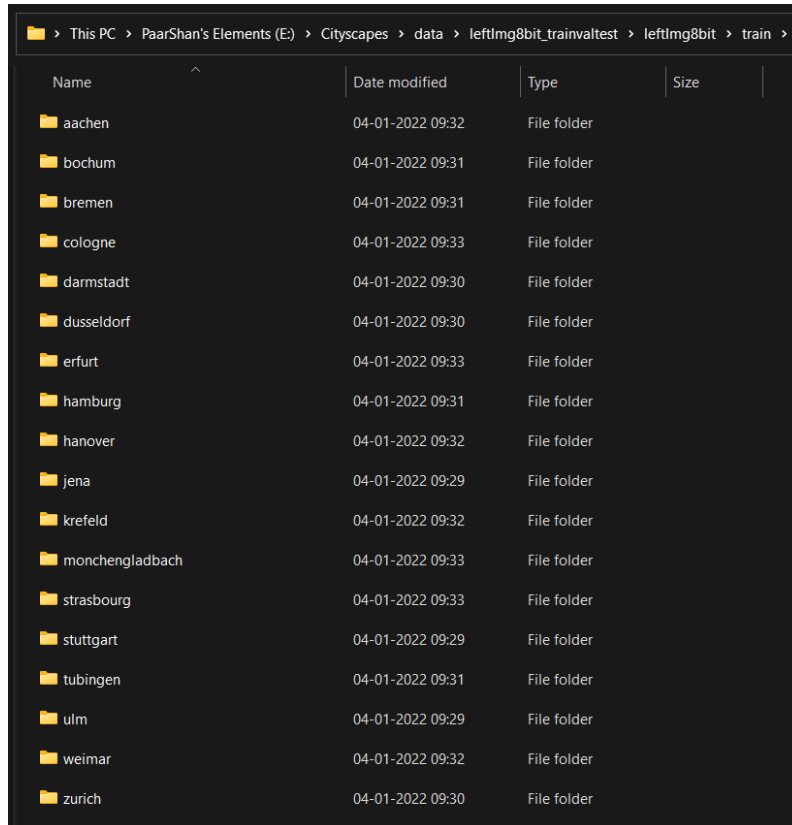
CHAPTER 4

IMPLEMENTATION AND RESULTS

4.1 Dataset Description

4.1.1 Cityscapes

Cityscapes is a dataset consisting of diverse urban street scenes across 50 different cities as shown in Figure 4.1 at varying times of the year as well as ground truths for several vision tasks including semantic segmentation, instance level segmentation, and stereo pair disparity inference. Cityscapes also provides coarse grain segmentation annotations. Cityscapes provides dense pixel level annotations for 5000 images at $1024 * 2048$ resolution as shown in Figure 4.2. Besides segmentation, cityscapes also provide stereo image pairs and ground truths for disparity inference tasks on both the normal and extra splits. It also consists of videos.



Name	Date modified	Type	Size
aachen	04-01-2022 09:32	File folder	
bochum	04-01-2022 09:31	File folder	
bremen	04-01-2022 09:31	File folder	
cologne	04-01-2022 09:33	File folder	
darmstadt	04-01-2022 09:30	File folder	
dusseldorf	04-01-2022 09:30	File folder	
erfurt	04-01-2022 09:33	File folder	
hamburg	04-01-2022 09:31	File folder	
hanover	04-01-2022 09:32	File folder	
jena	04-01-2022 09:29	File folder	
krefeld	04-01-2022 09:32	File folder	
monchengladbach	04-01-2022 09:33	File folder	
strasbourg	04-01-2022 09:33	File folder	
stuttgart	04-01-2022 09:29	File folder	
tubingen	04-01-2022 09:31	File folder	
ulm	04-01-2022 09:29	File folder	
weimar	04-01-2022 09:32	File folder	
zurich	04-01-2022 09:30	File folder	

Figure 4.1 – Different cities in Cityscapes dataset

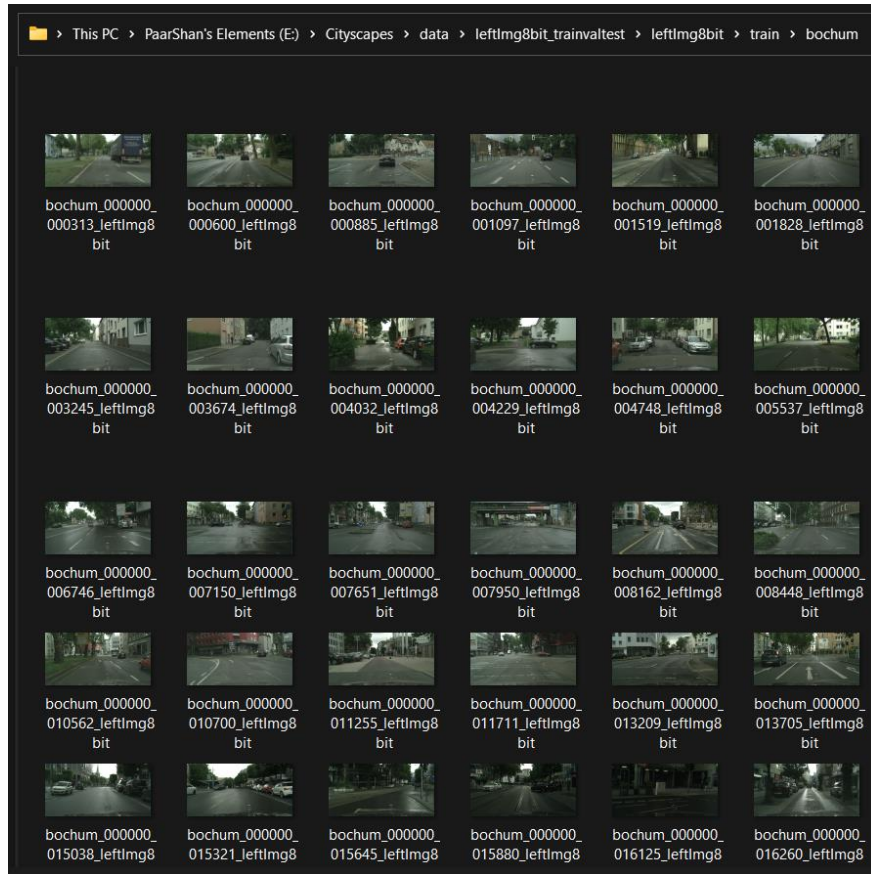


Figure 4.2 – A glimpse of Cityscapes dataset images

Some sample images from cityscapes dataset are shown in Figure 4.3.



Figure 4.3 – Cityscapes Dataset Sample Images

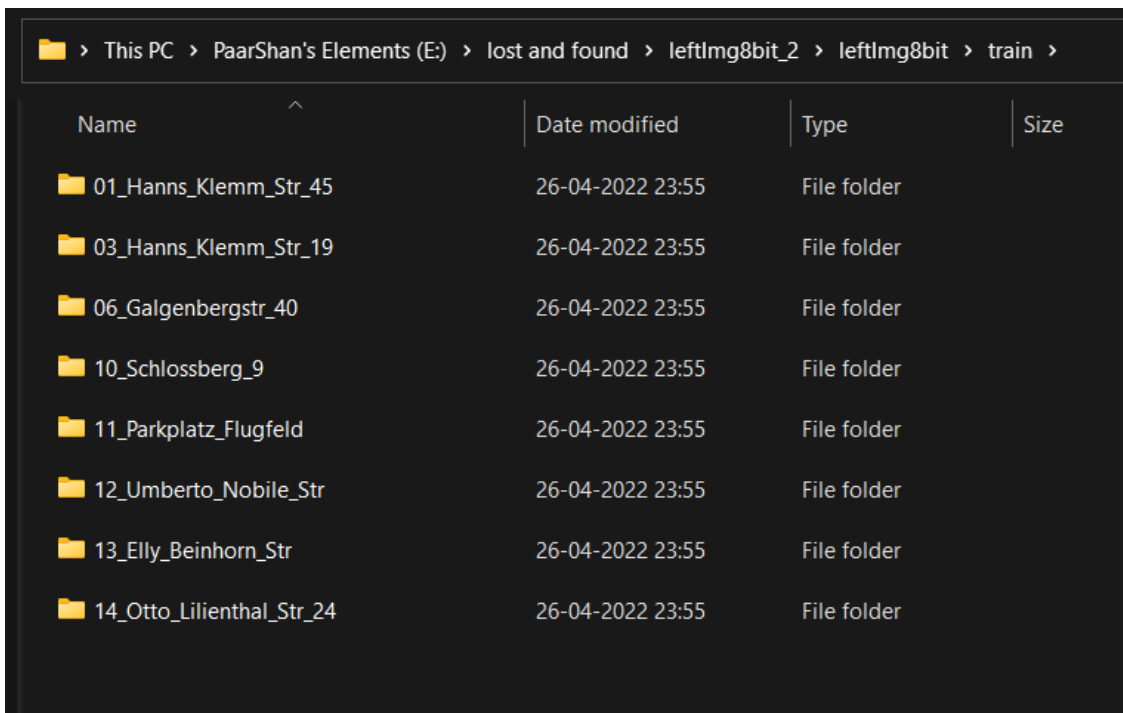
Source: <https://www.cityscapes-dataset.com/>

4.1.2 Lost and Found

The Lost and Found Dataset addresses the problem of detecting unexpected small obstacles on the road often caused by lost cargo. The dataset comprises 112 stereo video sequences with 2104 annotated frames (picking roughly every tenth frame from the recorded data). It has a resolution of 2048 x 1024.

The dataset is designed analogous to the 'Cityscapes' dataset as shown in Figure 4.4 and 4.5. The dataset provides:

- stereo image pairs in either 8 or 16 bit color resolution
- coarse semantic labels for objects and street



Name	Date modified	Type	Size
01_Hanns_Klemm_Str_45	26-04-2022 23:55	File folder	
03_Hanns_Klemm_Str_19	26-04-2022 23:55	File folder	
06_Galgenbergstr_40	26-04-2022 23:55	File folder	
10_Schlossberg_9	26-04-2022 23:55	File folder	
11_Parkplatz_Flugfeld	26-04-2022 23:55	File folder	
12_Umberto_Nobile_Str	26-04-2022 23:55	File folder	
13_Elly_Beinhorn_Str	26-04-2022 23:55	File folder	
14_Otto_Lilienthal_Str_24	26-04-2022 23:55	File folder	

Figure 4.4 – Different cities in Lost and Found dataset

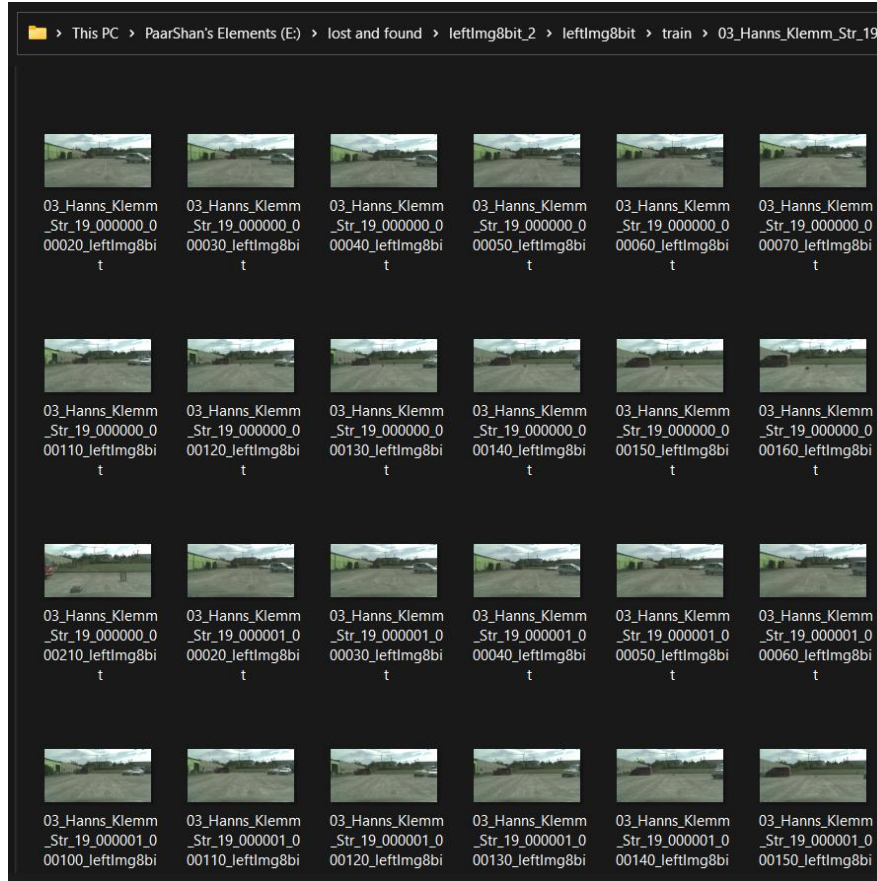


Figure 4.5 – A glimpse of Lost and Found dataset

Sample images from Lost and Found dataset are shown in Figure 4.6



Figure 4.6 – Lost and Found Dataset Sample Images

Source: <https://fishyscapes.com/dataset>

4.2 Implementation Details

The whole process of training and testing the presented network for building the model is executed under PyTorch framework with a memory of 16GB NVIDIA Quadro P5000.

4.2.1 Data Preprocessing

4.2.1.1 Disparity Map Generation

Disparity maps are used to calculate depth information from the two-dimensional images. The disparity maps are generated using left and right stereo images using Semi Global Matching algorithm. In a pair of images derived from stereo cameras, the apparent motion in pixels can be measured for every point and make an intensity image out of the measurements. Figure 4.7 shows a sample input and output for disparity map generation.

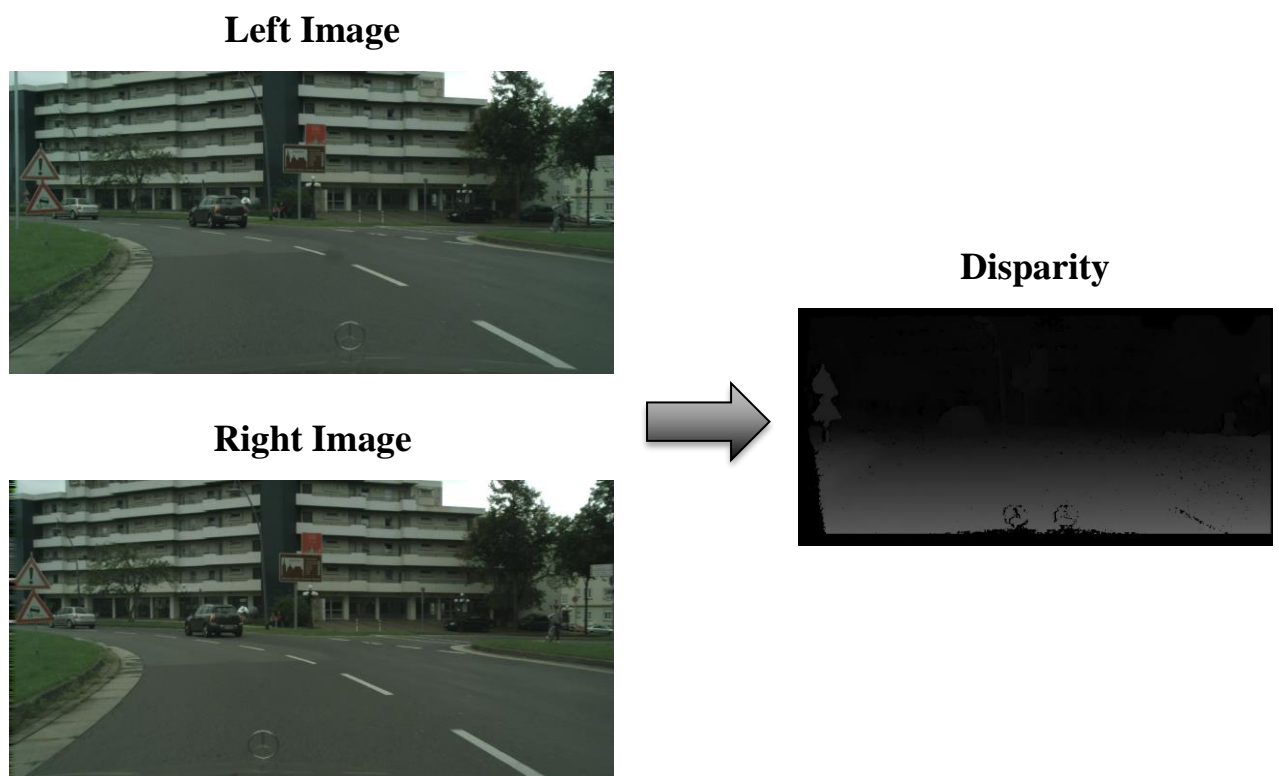


Figure 4.7 – Generating Disparity Maps

4.2.1.2 Data Preprocessing

i) Crop Black Area

Since the project uses left stereo image for segmentation, the extra information that it has on the left extreme when compared to the right stereo image has to be discarded. So, the extra black area is cropped as shown in Figure 4.8 during preprocessing.

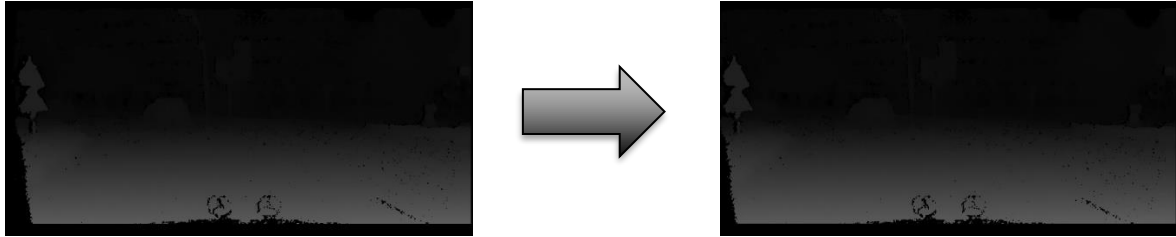


Figure 4.8 – Cropping Black Area

ii) Random Horizontal Flip

The obstacles that appear in only one side of the road may not be considered as an obstacle if they appear in another side. So, random horizontal flip is performed as shown in Figure 4.9 to counter this issue.

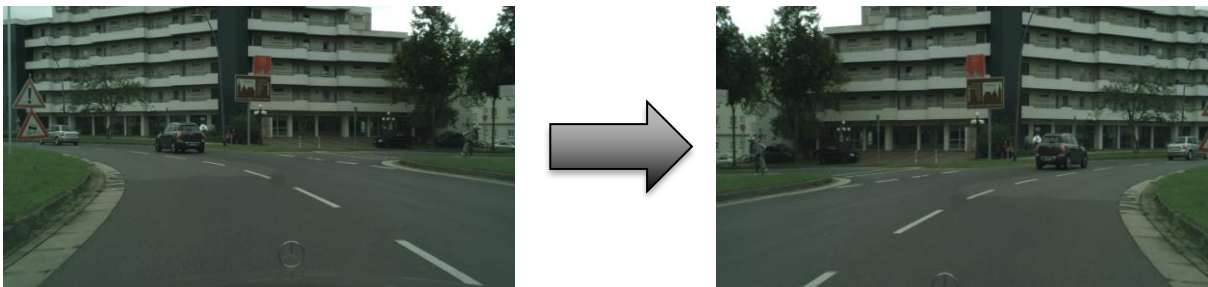


Figure 4.9 – Performing Random Horizontal Flip

iii) Random Scale Crop

Random cropping ensures that the same target will not always appear in the same position of the corresponding training sample image as shown in Figure 4.10, which effectively prevents the model from overfitting to the target spatial position.

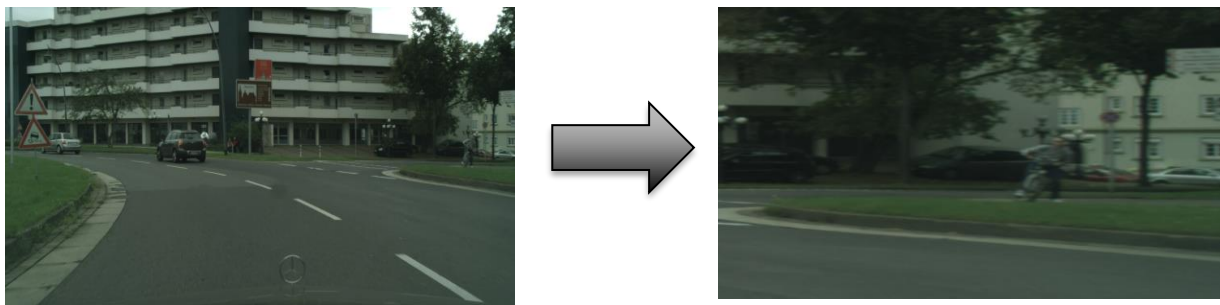


Figure 4.10 – Performing Random Scale Crop

iv) Random Gaussian Blur

Gaussian blurring is commonly used when reducing the size of an image. When downsampling an image, it is common to apply a low-pass filter as the one in Equation 4.1 to the image prior to resampling. It is used to soften sphere edges which often contain irregularities due to the rough surface of the marker and the output is similar to as shown in Figure 4.11.

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \quad (4.1)$$

where x is the distance from the origin in the horizontal axis,

y is the distance from the origin in the vertical axis,

σ is the standard deviation of the Gaussian distribution

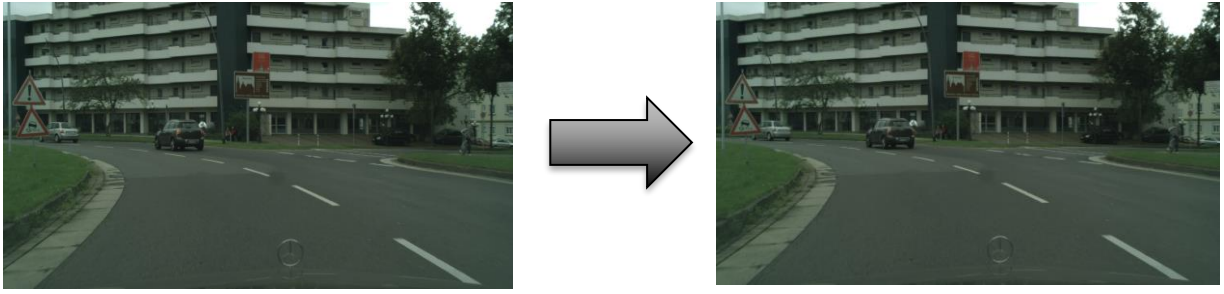


Figure 4.11 – Performing Gaussian Blur

4.2.1.3 Generating Train Labels

The cityscapes dataset images have around 32 classes of objects defined but, in this project, only 19 classes in cityscapes dataset and 1 (small obstacle) in Lost and Found dataset are segmented. The classes which are not segmented in this project are filtered out in this process of generating train labels similar to one shown in Figure 4.12.

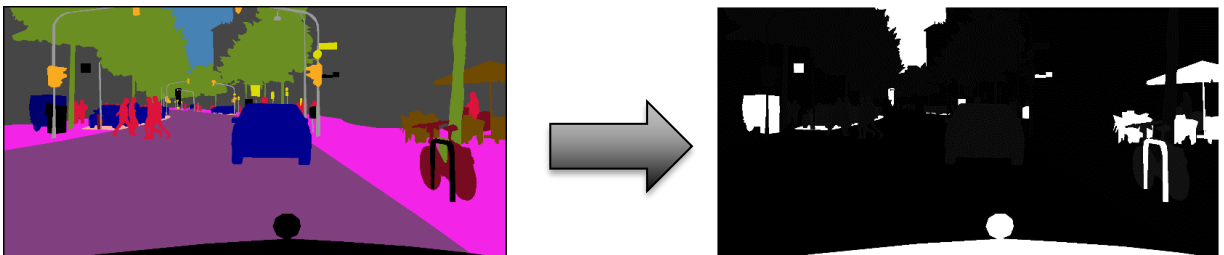


Figure 4.12 – Generating Train Labels

4.2.2 Model Definition

4.2.2.1 RGB Layer

Here, features from the RGB input are extracted and then fed to the AFC module for further fusion with depth features and learning. Figure 4.13 shows the definition of RGB layer and Figure 4.14 shows the output feature maps.

```
(layer1): Sequential(
  (0): BasicBlock(
    (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
    (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (1): BasicBlock(
    (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
    (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
)
```

Figure 4.13 – Defining RGB Layer of RFNet

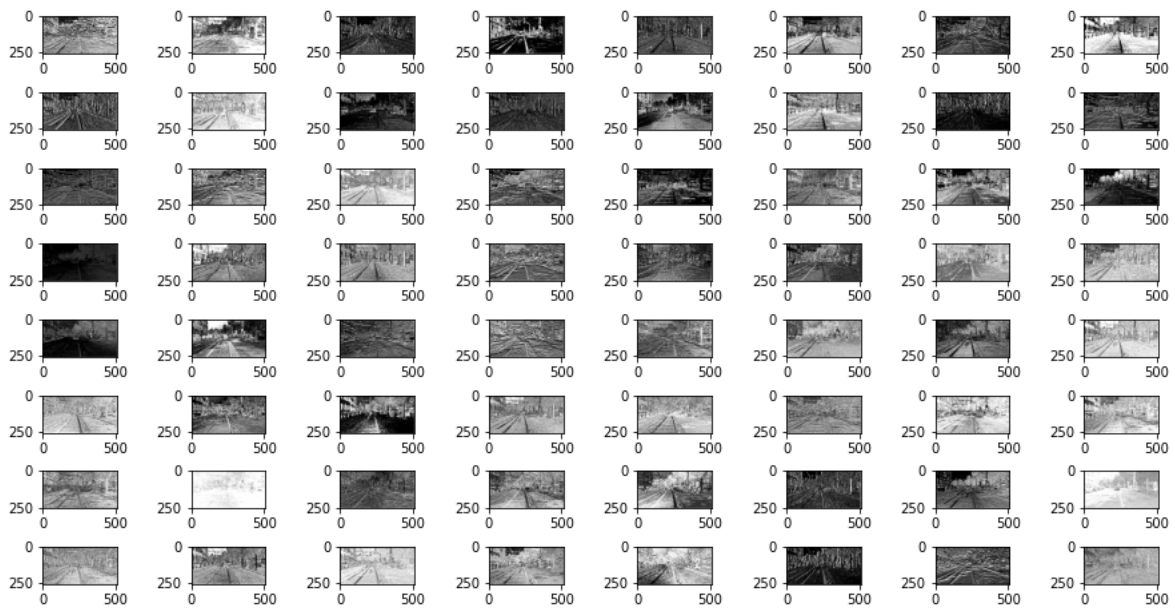


Figure 4.14 – Extracted Features from RGB Layer 1

4.2.2.2 Depth Layer

Here, features from the Depth input are extracted and then fed to the AFC module for further fusion with RGB features and learning. Figure 4.15 shows the definition of Depth layer and Figure 4.16 shows the output feature maps.

```
(layer1_d): Sequential(
  (0): BasicBlock(
    (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
    (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (1): BasicBlock(
    (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
    (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
)
```

Figure 4.15 – Defining Depth Layer of RFNet

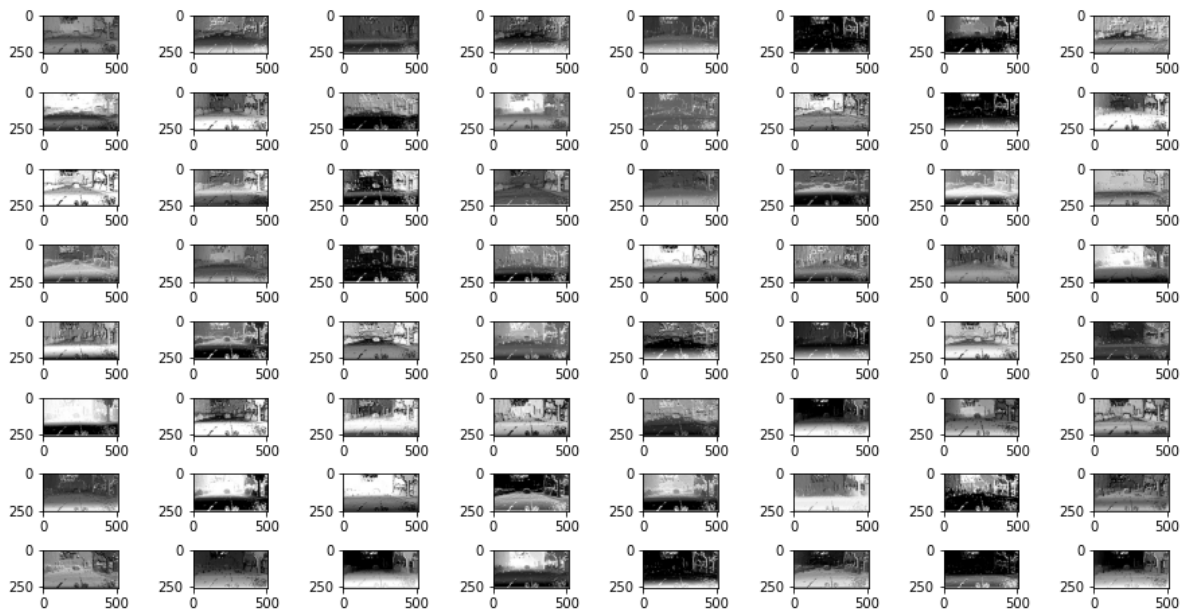


Figure 4.16 – Extracted Features from Depth Layer 1

4.2.2.3 Attention Feature Complementary

Attention Feature Complementary module is used to make the network focus on learning more complementary informative features from RGB and Depth branches. Once the feature extraction is done, Squeeze and Excitation is performed after which the features of both RGB and depth inputs are element-wise added which acts as the input for the next block of RGB branch. AFC module makes use of cross model information from both RGB and Depth inputs. In the AFC module, we leverage the SE block as the channel attention module. Figure 4.17 shows the definition of AFC Module and Figure 4.18 shows the output feature maps.

```
(attention_1): Sequential(
  (0): AdaptiveAvgPool2d(output_size=1)
  (1): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1))
  (2): Sigmoid()
)
(attention_1_d): Sequential(
  (0): AdaptiveAvgPool2d(output_size=1)
  (1): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1))
  (2): Sigmoid()
)
```

Figure 4.17 – Defining Attention Feature Complementary Module

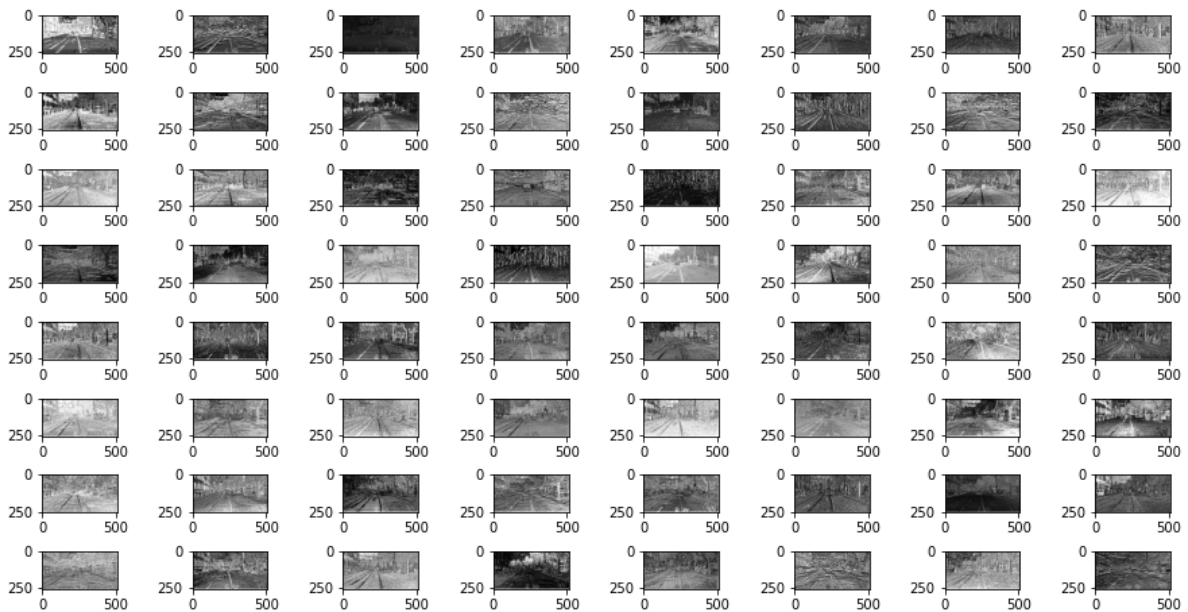


Figure 4.18 – Extracted Features from AFC 1

4.2.2.4 Spatial Pyramid Pooling

The fused feature maps include rich high-level semantic information after four ResNet blocks and one AFC module. We use Spatial Pyramid Pooling to average features over aligned grids with varied granularities before upsampling, to increase the receptive field to cover pixels of huge objects while retaining real-time speed. It gives output of fixed length of 128 channels irrespective of the input dimensions. Figure 4.19 shows the definition of SPP Module.

```
(spp): SpatialPyramidPooling(  
  (spp): Sequential(  
    (spp_bn): _BNReLUConv(  
      (norm): BatchNorm2d(512, eps=1e-05, momentum=0.005, affine=True, track_running_stats=True)  
      (relu): ReLU(inplace=True)  
      (conv): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)  
    )  
    (spp0): _BNReLUConv(  
      (norm): BatchNorm2d(128, eps=1e-05, momentum=0.005, affine=True, track_running_stats=True)  
      (relu): ReLU(inplace=True)  
      (conv): Conv2d(128, 42, kernel_size=(1, 1), stride=(1, 1), bias=False)  
    )  
    (spp1): _BNReLUConv(  
      (norm): BatchNorm2d(128, eps=1e-05, momentum=0.005, affine=True, track_running_stats=True)  
      (relu): ReLU(inplace=True)  
      (conv): Conv2d(128, 42, kernel_size=(1, 1), stride=(1, 1), bias=False)  
    )  
    (spp2): _BNReLUConv(  
      (norm): BatchNorm2d(128, eps=1e-05, momentum=0.005, affine=True, track_running_stats=True)  
      (relu): ReLU(inplace=True)  
      (conv): Conv2d(128, 42, kernel_size=(1, 1), stride=(1, 1), bias=False)  
    )  
    (spp_fuse): _BNReLUConv(  
      (norm): BatchNorm2d(254, eps=1e-05, momentum=0.005, affine=True, track_running_stats=True)  
      (relu): ReLU(inplace=True)  
      (conv): Conv2d(254, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)  
    )  
  )  
)
```

Figure 4.19 – Defining Spatial Pyramid Pooling Module

4.2.2.5 Upsampling

The decoder upsamples semantically rich visual features in coarse spatial resolution to the input resolution. A simple decoder is used here, with three simple upsampling modules and skip connections from the encoder. Figure 4.20 shows the definition of Upsampling Module and Figure 4.21 shows the output feature maps.

```

(upsample): ModuleList(
  (0): _Upsample(
    (bottleneck): _BNReLUConv(
      (norm): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
    )
    (blend_conv): _BNReLUConv(
      (norm): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    )
  )
)

```

Figure 4.20 – Defining Upsampling

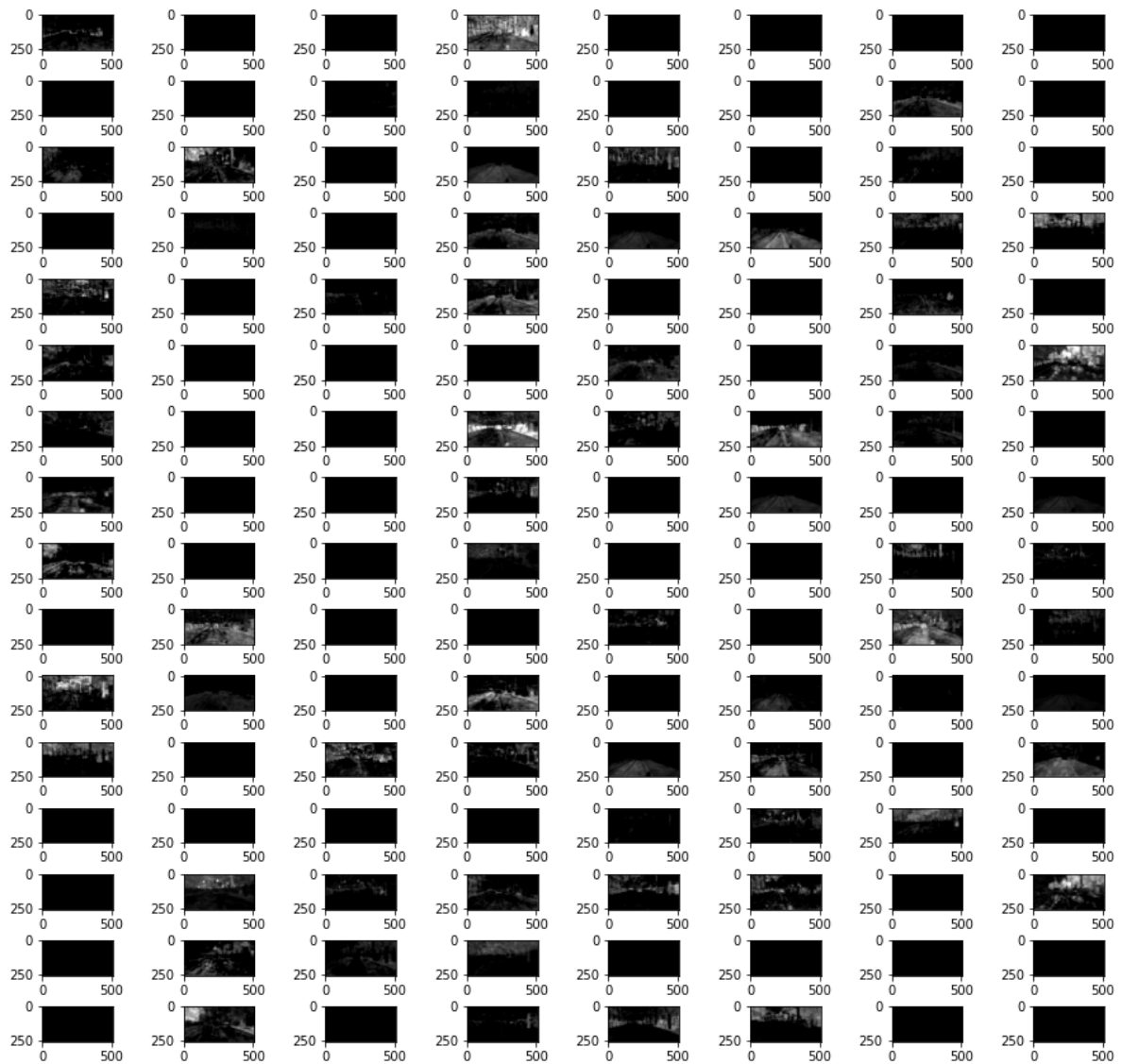


Figure 4.21 – Extracted Features from Upsampling 3

4.2.3 Training and Validation

4.2.3.1 Loading Images

3652 set of images (RGB, disparity, generated train label) and 1529 set of images for training and validation respectively are loaded as shown in Figure 4.22.

```
Found 3652 train RGB images
Found 3652 train disparity images
Found 1529 val RGB images
Found 1529 val disparity images

Starting Epoch: 0
Total Epoches: 200
```

Figure 4.22 – Loading images and initializing epochs

4.2.3.2 Training

0th and 1st epoch training

Figure 4.23 shows the training process for first two epochs and its loss.

```
Train loss: 0.184: 100%|██████████| 457/457 [06:05<00
:00, 1.25it/s]
[Epoch: 0, numImages: 3652]
Loss: 84.088
 0%|          | 0/457 [00:00<?, ?it/s]
=>Epochs 1, learning rate = 0.0001,
previous best = 0.0000
Train loss: 0.139: 100%|██████████| 457/457 [06:05<00
:00, 1.25it/s]
: 0%|          | 0/510 [00:00<?, ?it/s][Epoch: 1,
numImages: 3652]
Loss: 63.735
```

Figure 4.23 – 0th and 1st epoch training with loss

Validation

```
Test loss: 0.362: 100%|██████████| 510/510 [09:36<00:
00, 1.13s/it]
```


Figure 4.24 shows the accuracy and IoU scores of each class for the first two epochs.

Accuracy of each class		IOU of each class	
-----Acc of each classes-----		-----IoU of each classes-----	
road	: 71.734687 %	road	: 70.796412 %
sidewalk	: 45.425221 %	sidewalk	: 22.182159 %
building	: 78.873478 %	building	: 72.259015 %
wall	: 19.258400 %	wall	: 9.962563 %
fence	: 32.292092 %	fence	: 21.413850 %
pole	: 74.257493 %	pole	: 28.580116 %
traffic light	: 59.973246 %	traffic light	: 22.751335 %
traffic sign	: 56.029549 %	traffic sign	: 39.687030 %
vegetation	: 88.974214 %	vegetation	: 80.003273 %
terrain	: 53.443259 %	terrain	: 25.999952 %
sky	: nan %	sky	: nan %
person	: 85.926386 %	person	: 14.048063 %
rider	: 0.000000 %	rider	: 0.000000 %
car	: 70.184027 %	car	: 16.920188 %
truck	: 29.372440 %	truck	: 7.348440 %
bus	: 2.829125 %	bus	: 2.588400 %
train	: 0.000000 %	train	: 0.000000 %
motorcycle	: 0.000000 %	motorcycle	: 0.000000 %
bicycle	: 50.164395 %	bicycle	: 15.188648 %
small obstacles	: 13.308440 %	small obstacles	: 12.676061 %

Figure 4.24 – Accuracy and IoU of each class

The performance metrics after first validation is shown in Figure 4.25.

```

Validation:
[Epoch: 1, numImages: 4074]
Acc:0.7218723947829357, Acc_class:0.
43791918503402166, mIoU:0.24337131868446055, fwIoU:
0.6546106621927194, Recall: 0.43791918503402166,
Precision: 0.3516263689225144, F1 score: 0.
3900571211968257

```

Figure 4.25 – First validation with performance metrics

Last two epochs

Figure 4.26 shows the training process for last two epochs and its loss.

```
=>Epochs 198, learning rate = 0.0000
, previous best = 0.6641
Train loss: 0.029: 100%|██████████| 457/457 [06:04<
00:00, 1.25it/s]
[Epoch: 198, numImages: 3652]
Loss: 13.333
0%|██████████| 0/457 [00:00<?, ?it/s]
=>Epochs 199, learning rate = 0.0000
, previous best = 0.6641
Train loss: 0.030: 100%|██████████| 457/457 [06:04<
00:00, 1.25it/s]
[Epoch: 199, numImages: 3652]
Loss: 13.854
```

Figure 4.26 – Last two epoch training with loss

Figure 4.27 shows the accuracy and IoU scores of each class after all epochs.

Accuracy of each class		IOU of each class	
-----Acc of each classes-----		-----IoU of each classes-----	
road	: 93.249238 %	road	: 93.092738 %
sidewalk	: 92.184425 %	sidewalk	: 39.019288 %
building	: 92.367309 %	building	: 89.599634 %
wall	: 58.449191 %	wall	: 43.443118 %
fence	: 75.523616 %	fence	: 56.599351 %
pole	: 84.283815 %	pole	: 58.306479 %
traffic light	: 91.414418 %	traffic light	: 57.989478 %
traffic sign	: 89.794620 %	traffic sign	: 68.848205 %
vegetation	: 94.031824 %	vegetation	: 89.290858 %
terrain	: 74.565139 %	terrain	: 49.397314 %
sky	: nan %	sky	: nan %
person	: 94.920881 %	person	: 75.019061 %
rider	: 75.433845 %	rider	: 52.848618 %
car	: 96.706293 %	car	: 90.902394 %
truck	: 76.683269 %	truck	: 60.410373 %
bus	: 85.706044 %	bus	: 76.177754 %
train	: 85.728533 %	train	: 69.723084 %
motorcycle	: 67.415097 %	motorcycle	: 50.379801 %
bicycle	: 89.585653 %	bicycle	: 66.806569 %
small obstacles	: 75.384655 %	small obstacles	: 65.520726 %

Figure 4.27 – Accuracy and IoU of each class

The performance metrics after all validations are shown in Figure 4.28.

```
Validation:
[Epoch: 199, numImages: 4074]
Acc:0.9264008228438746, Acc_class:0.
8386462451786334, mIoU:0.6596709700067906, fwIoU: 0
.8873225651448046, Recall: 0.8386462451786334,
Precision: 0.7489150097561362, F1 score: 0.
7912447585094406
```

Figure 4.28 – Last validation with performance metrics

4.2.3.3 Hyperparameters for Training

The hyperparameters used for the training process are shown in Table 4.1.

Table 4.1 – Hyperparameters for training

Hyperparameter	Value
Learning Rate	0.0004
Optimizer	Adam
Number of Iterations	200
Loss	Cross Entropy Loss
Weight Loss	$1 * 10^{-4}$
Batch Size	8
Save Steps	10
Activation Function	ReLu

4.2.3.4 Evaluation

1529 set of images (RGB, disparity, generated train label) for evaluation/testing are loaded shown in Figure 4.29.

```

Found 1529 val RGB images
Found 1529 val disparity images

: 0%|          | 1/1529 [00:11<4:48:06, 11.31s/it]
Forward time per img (batch size=1): 0.070 (Mean: 0.
070)
: 0%|          | 2/1529 [00:12<2:12:27, 5.20s/it]
Forward time per img (batch size=1): 0.060 (Mean: 0.
065)
: 0%|          | 3/1529 [00:13<1:23:11, 3.27s/it]
Forward time per img (batch size=1): 0.070 (Mean: 0.
067)
: 100%|████████| 1528/1529 [24:52<00:00, 1.04it/
s]Forward time per img (batch size=1): 0.070 (Mean:
0.066)
: 100%|████████| 1529/1529 [24:54<00:00, 1.02it/
s]

```

Figure 4.29 – Loading images and performing evaluation/testing

Figure 4.30 shows the accuracy and IoU scores of each class after evaluation.

Accuracy of each class		IOU of each class	
-----Acc of each classes-----		-----IoU of each classes-----	
road	: 92.366566 %	road	: 48.375056 %
sidewalk	: 11.194995 %	sidewalk	: 10.444095 %
building	: 91.108843 %	building	: 88.130307 %
wall	: 57.394097 %	wall	: 43.890775 %
fence	: 75.255932 %	fence	: 57.461130 %
pole	: 83.425929 %	pole	: 59.129200 %
traffic light	: 90.969410 %	traffic light	: 58.573492 %
traffic sign	: 89.512352 %	traffic sign	: 69.411340 %
vegetation	: 93.700831 %	vegetation	: 89.397769 %
terrain	: 75.868464 %	terrain	: 48.692373 %
sky	: nan %	sky	: nan %
person	: 94.740084 %	person	: 75.679641 %
rider	: 74.184484 %	rider	: 53.237028 %
car	: 96.262670 %	car	: 91.131754 %
truck	: 82.006618 %	truck	: 63.901050 %
bus	: 88.679418 %	bus	: 76.387834 %
train	: 89.085928 %	train	: 63.296533 %
motorcycle	: 69.260652 %	motorcycle	: 50.710353 %
bicycle	: 89.506118 %	bicycle	: 67.083073 %
small obstacles	: 81.643416 %	small obstacles	: 19.420129 %

Figure 4.30 – Accuracy and IoU of each class

The performance metrics after evaluation are shown in Figure 4.31.

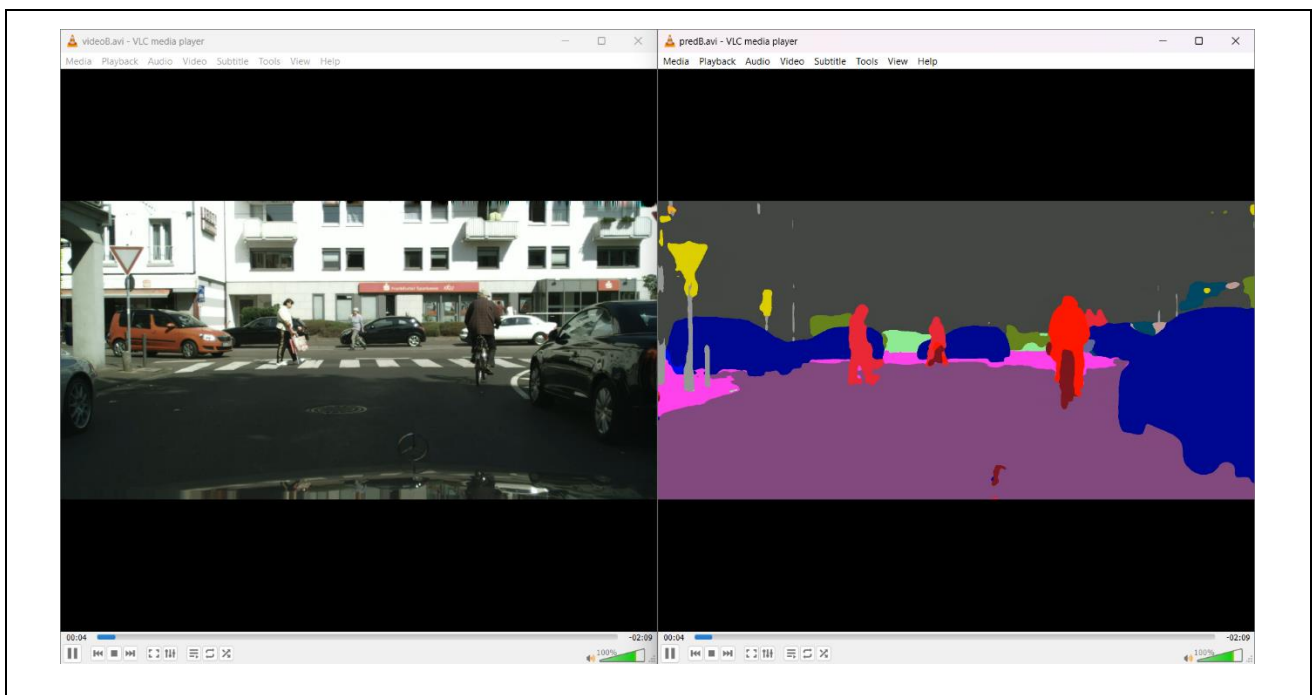
```
Acc:0.6392188868240457, Acc_class:0.
6392188868240457, mIoU:0.5970278584789688, fwIoU: 0
.46815200208580704, Recall: 0.8032456872929111,
Precision: 0.6979367825809281, F1 score: 0.
7468974916264979
```

Figure 4.31 – Performance metrics for evaluation/testing

4.2.3.5 Video Segmentation

Here, the video is split into frames and then fed into the exported trained model to segment the obstacles. Table 4.2 shows a sequence of inputs and their corresponding outputs.

Table 4.2 – Video frames and its corresponding prediction

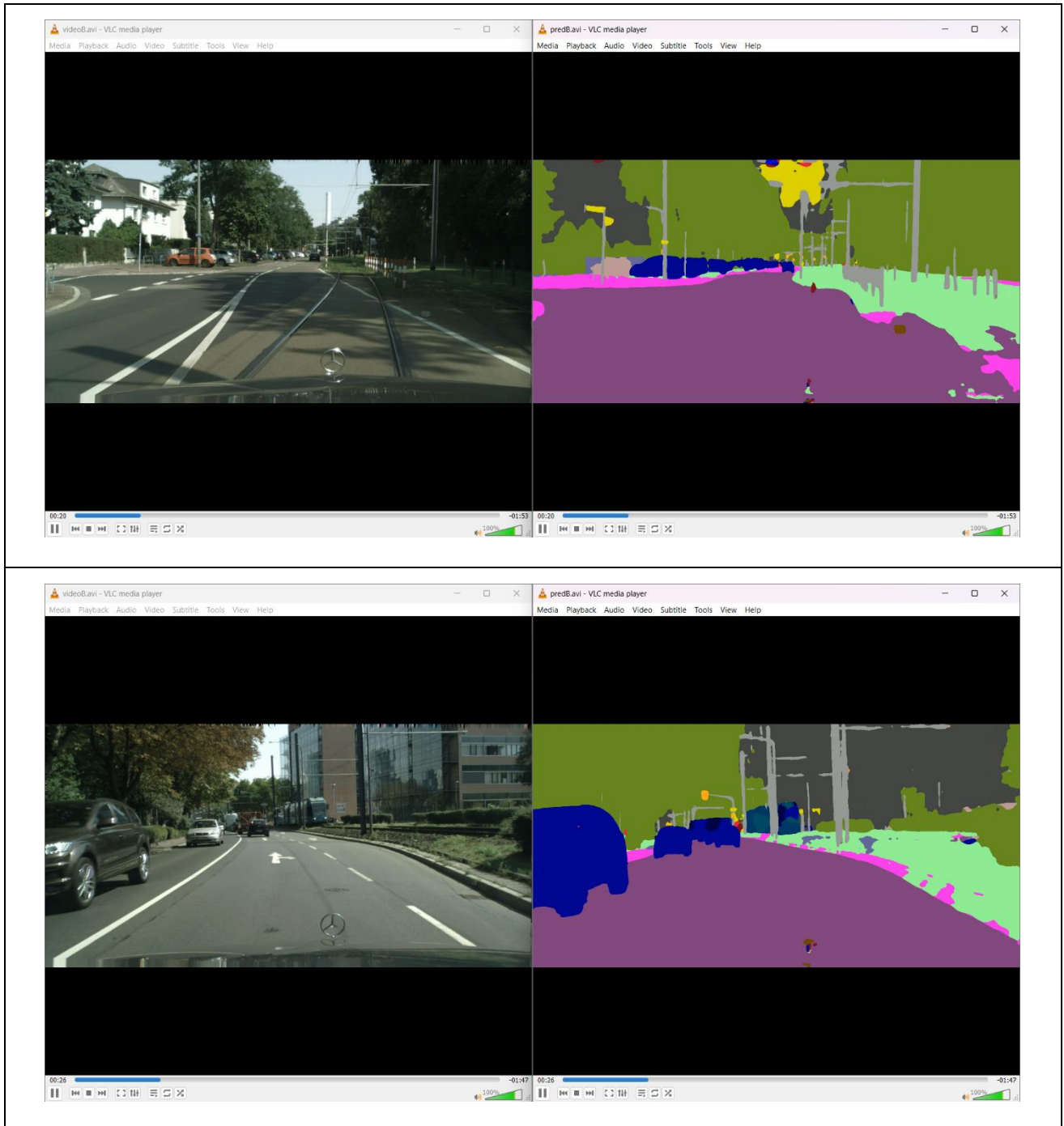


Continued on next page

Continued from previous page



Continued on next page



4.3 Results and Test Cases


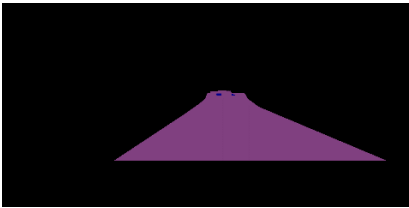
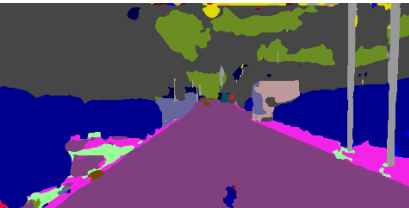
The different colors shown in the Figure 4.32 represent the different types of objects that can be considered as obstacles.

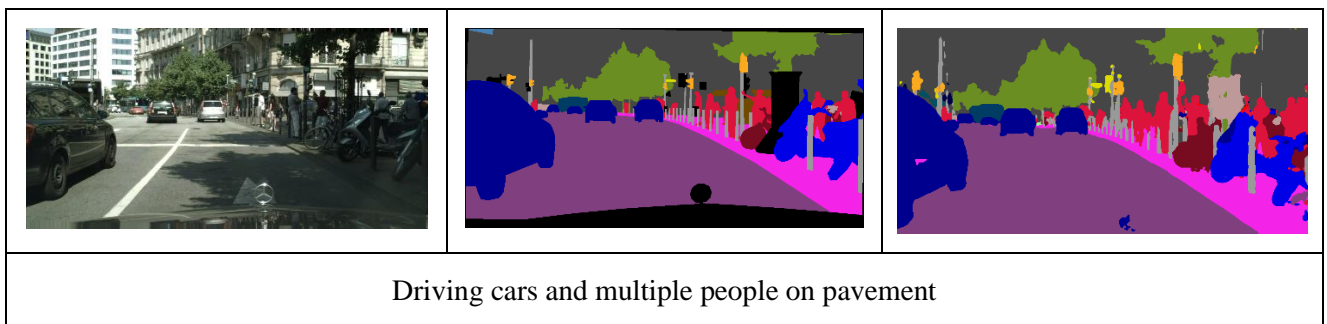
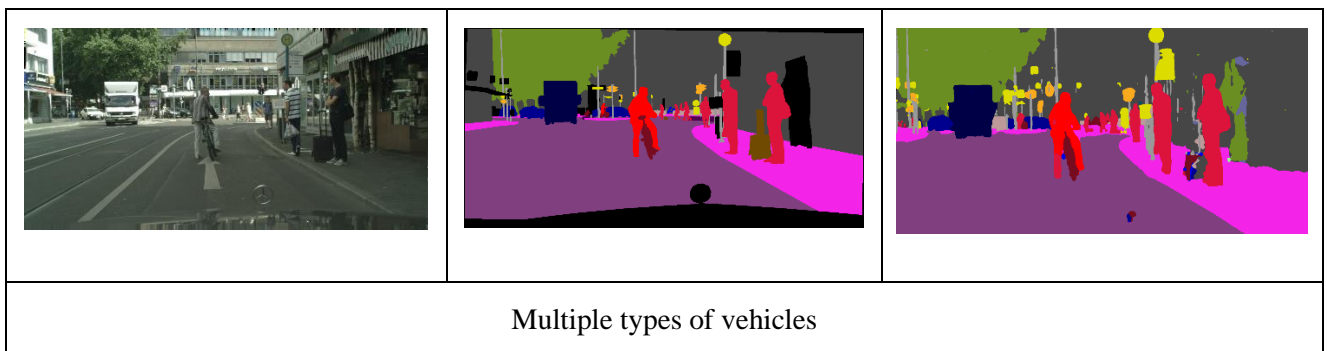
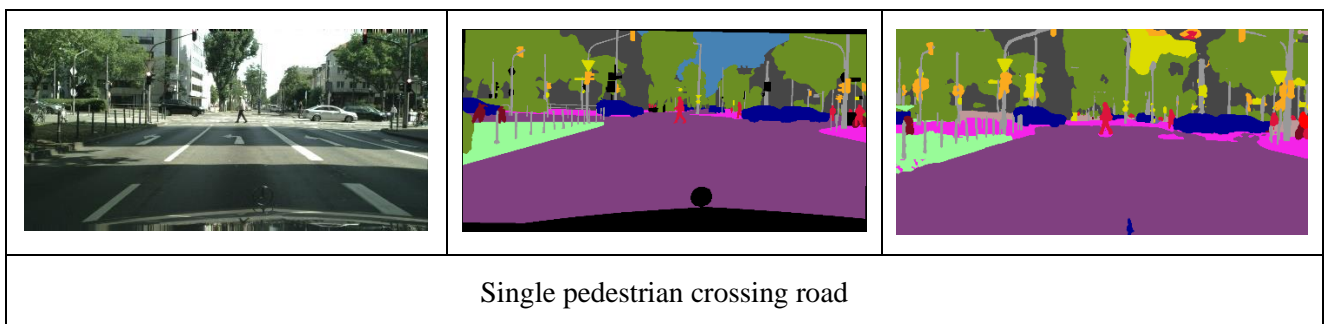
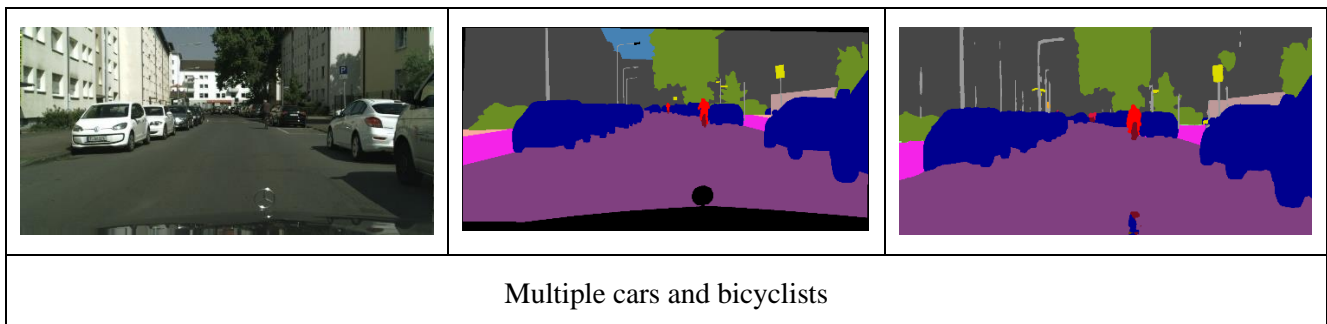
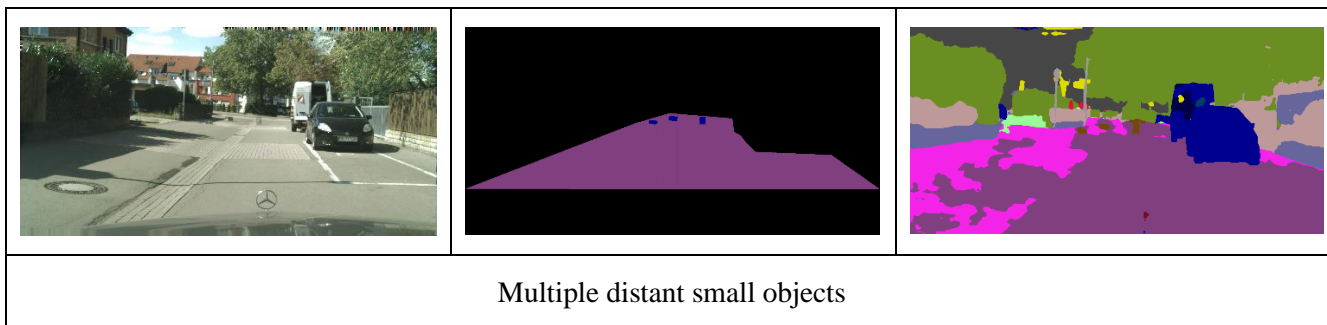
LEGEND			
	Road		Pedestrian
	Sidewalk		Rider
	Building		Car
	Wall		Truck
	Fence		Bus
	Pole		Train
	Sign Board		Motorcycle
	Traffic Light		Bicycle
	Vegetation		Small obstacles
	Terrain		

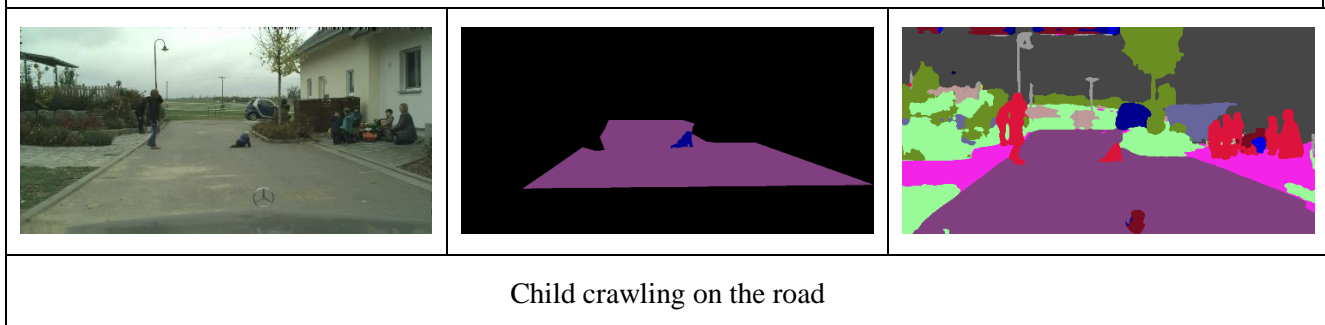
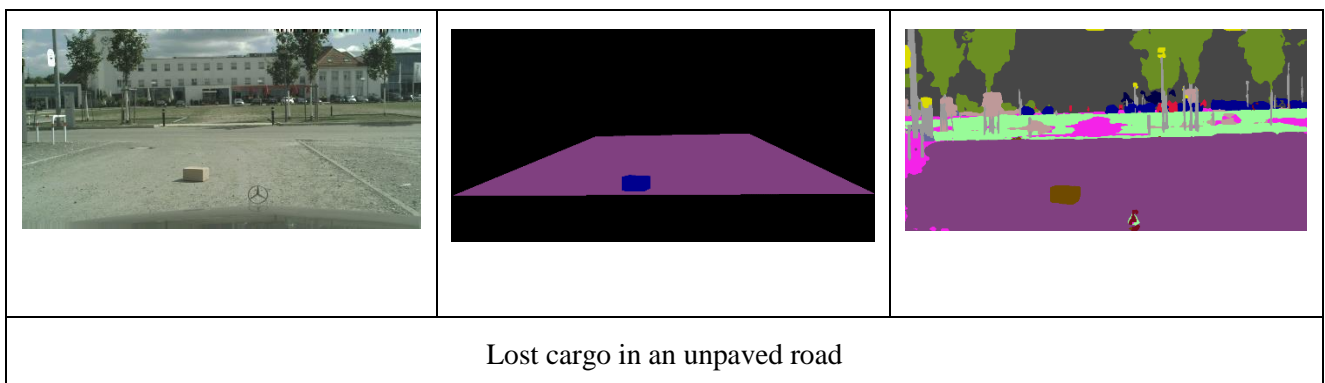
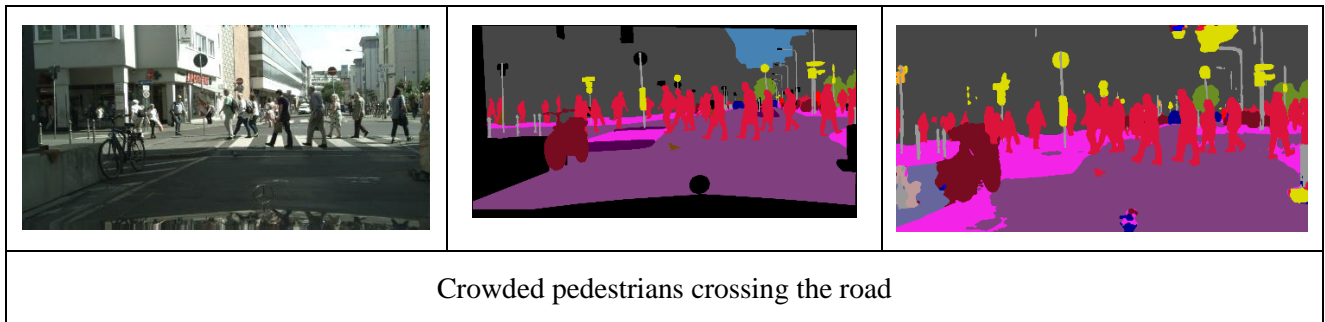
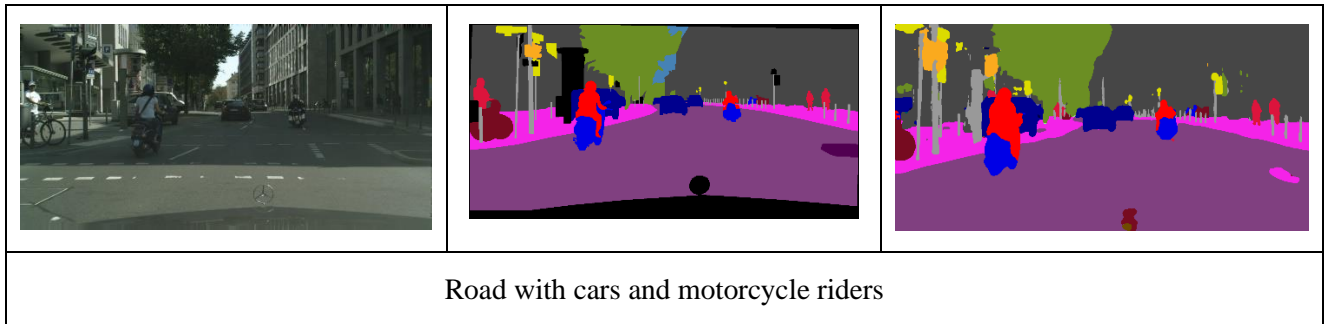
Figure 4.32 – Legend for obstacles with corresponding color

Table 4.3 shows a list of possible test cases along with the ground truth and the model's prediction.

Table 4.3 – Possible test cases with description

Image	Actual Output	Prediction
		
Single distant small object		





4.4 Performance Metrics

Mean Intersection-Over-Union(mIoU)

Mean Intersection-Over-Union mentioned in Equation 4.3 is a common evaluation metric for semantic image segmentation, which first computes the IOU for each semantic class and then computes the average over classes. IOU is defined as follows:

$$IOU_{obstacle} = TP_{obstacle} / (TP_{obstacle} + FP_{obstacle} + FN_{obstacle}) \quad (4.2)$$

$$mIoU = \Sigma IOU_{obstacle} / num_obstacle \quad (4.3)$$

In terms of area,

$$IOU = Area\ of\ Overlap / Area\ of\ Union \quad (4.4)$$

The predictions are accumulated in a confusion matrix, weighted by sample weight and the metric are then calculated from it.

Frequency weighted IOU

The frequency-weighted intersection over union (FWIoU) is an extension of MIOU mentioned in Equation 4.4 in which weights are assigned according to the frequency of each class.

$$weight = no.\ of\ obstacles\ of\ a\ class / total\ no.\ of\ obstacles \quad (4.5)$$

$$fwIoU = \Sigma IOU_{obstacle} * weight / num_obstacle \quad (4.6)$$

Pixel Accuracy

It is the percent of pixels in your image that are classified correctly

$$Pixel\ Accuracy = \frac{\#TP + \#TN}{\#TP + \#TN + \#FP + \#FN} \quad (4.7)$$

Precision

Precision quantifies the number of positive class predictions that actually belong to the positive class. In other words, it is the ability of a classification model to identify only the relevant data points.

$$precision = \frac{tp}{tp + fp} \quad (4.8)$$

Recall

The recall is calculated as the ratio between the numbers of Positive samples correctly classified as Positive to the total number of Positive samples. The recall measures the model's ability to detect positive samples. The higher the recall, the more positive samples detected.

$$Recall = \frac{TruePositive}{TruePositive + FalseNegative} \quad (4.9)$$

F1 Score

The F1-score score sums up the predictive performance of a model by combining two otherwise competing metrics, precision and recall. It is primarily used to compare the performance of two classifiers.

$$\begin{aligned} F1 &= \frac{2}{recall^{-1} + precision^{-1}} = 2 \frac{precision \cdot recall}{precision + recall} \\ &= \frac{tp}{tp + \frac{1}{2}(fp + fn)} \end{aligned} \quad (4.10)$$

4.5 Performance Graphs

Figures 4.33 - 4.41 shows the various graphs for performance metrics like validation accuracy, mean intersection over union, frequency weighted mean intersection over union, precision, recall, F1 score and also for training and validation loss.

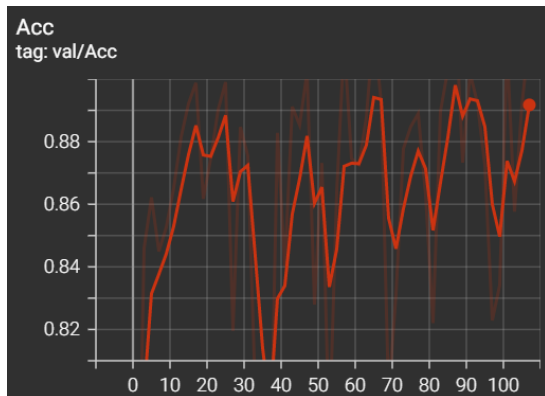


Figure 4.33 – Validation Accuracy Graph

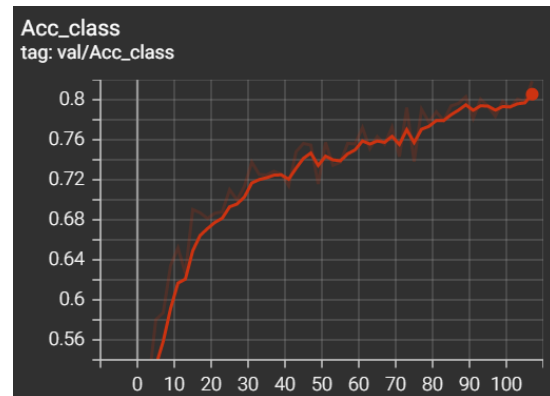


Figure 4.34 – Class Accuracy Graph

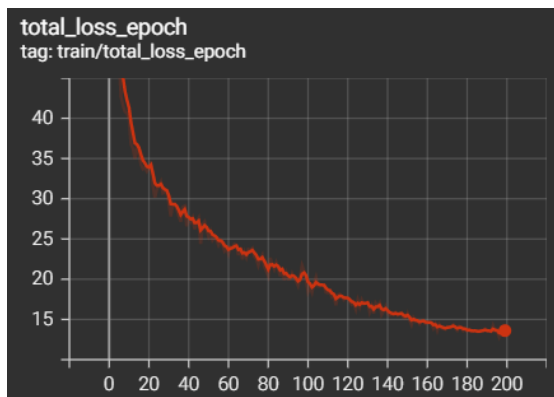


Figure 4.35 – Training Loss Graph

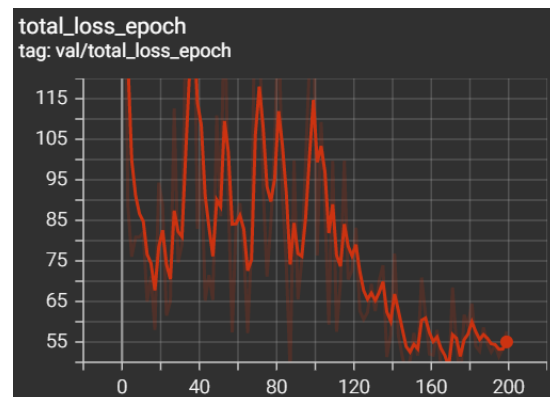


Figure 4.36 – Validation Loss Graph

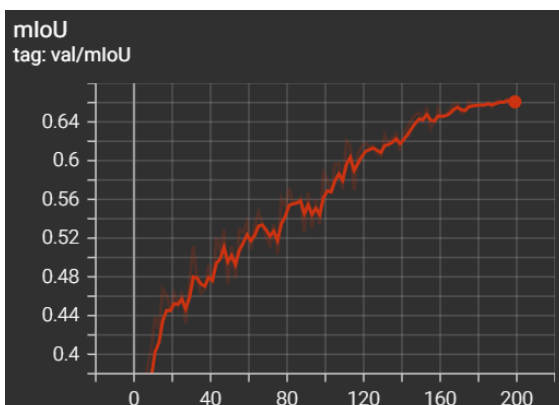


Figure 4.37 – Mean Intersection Over Union Graph

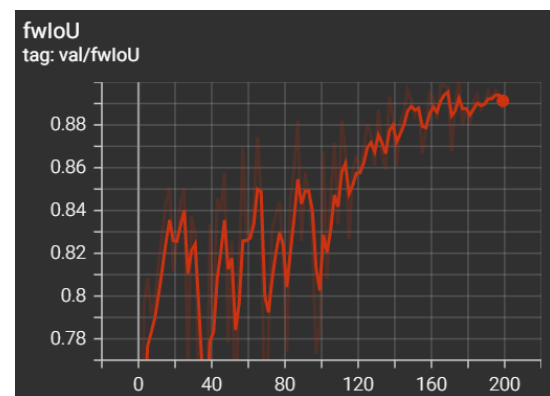


Figure 4.38 – Frequency Weight Mean Intersection Over Union Graph

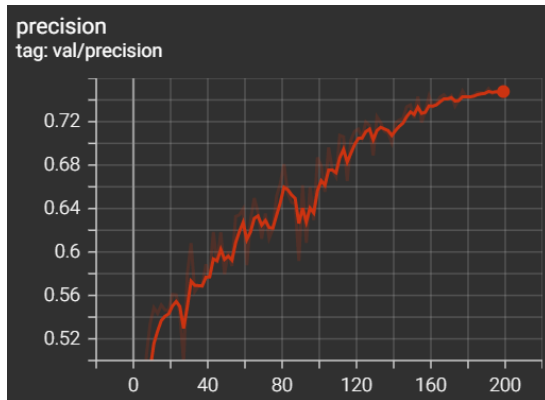


Figure 4.39 – Precision Graph

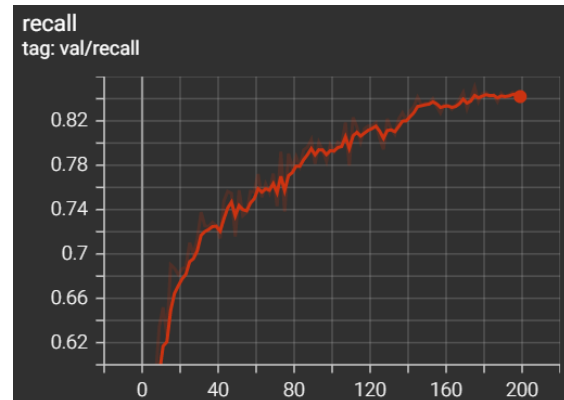


Figure 4.40 – Recall Graph

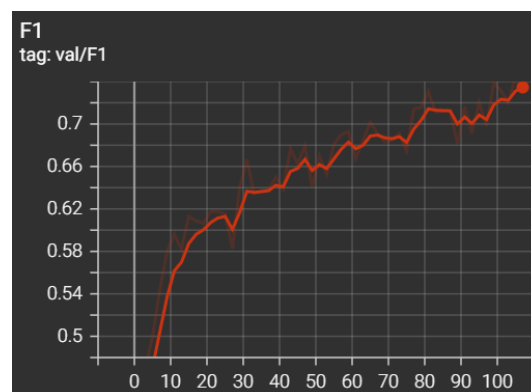


Figure 4.41 – F1 Score Graph

Table 4.4 lists out the scores for various performance metrics.

Table 4.4 – Performance metrics

Performance Metric	Score
mIoU	59.7%
fwIoU	46.8%
Accuracy	63.9%
Precision	69.7%
Recall	80.3%
F1 Score	0.74

4.6 Comparative Analysis

The Mean Intersection Over Union values of different models are compared in the Figure 4.42 and Table 4.5.

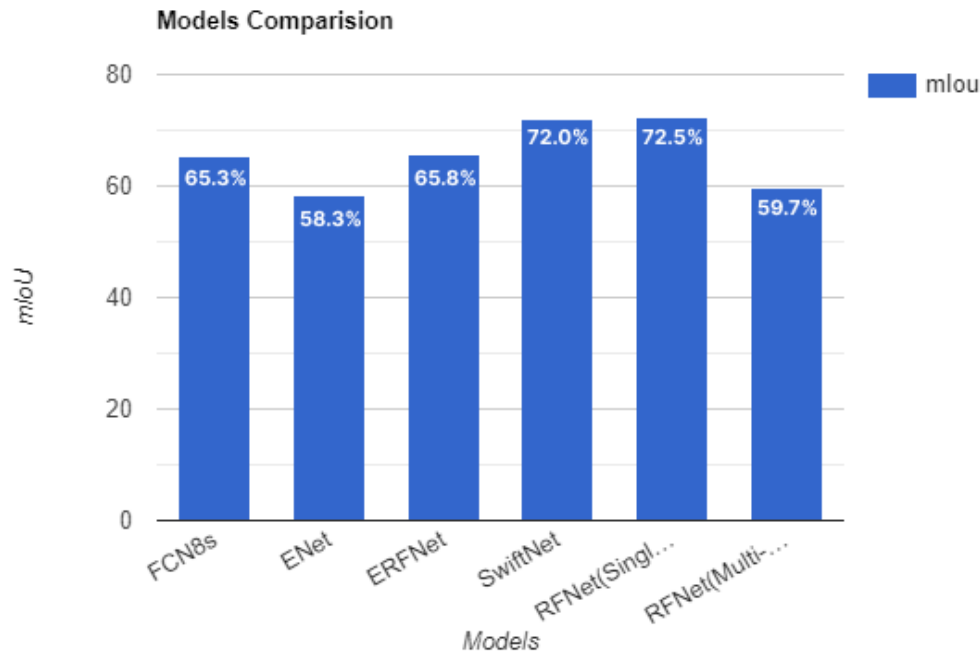


Figure 4.42 – Different models comparison of mIoU value

Table 4.5 – Comparison of semantic segmentation methods on validation set

Network	mIoU (%)
FCN8s	65.3%
ENet	58.3%
ERFNet	65.8%
SwiftNet	72.0%
RFNet	72.5% (Single Dataset) 59.7% (Multi Dataset)

The RFNet was tested for the mIoU performance metric with multiple datasets (Cityscapes and Lost and Found).

CHAPTER 5

CONCLUSION AND FUTURE EXPANSION

5.1 Conclusion

An end-to-end model has been introduced in this work. The RFNet network is proposed here as a real-time fusion network for RGB-D semantic segmentation on road-driving images. RFNet successfully uses complementary depth information with the intended AFC module, considerably improving accuracy over simply RGB-based systems. RFNet can detect unexpected minor obstacles using the described multi-source training technique, enriching the identifiable classes required to tackle the problem.

5.2 Future Work

This work will serve as the base for autonomous driving when combined with other works like Sign Board Recognition, Traffic Light Recognition etc. Currently we have trained our model for Cityscapes and Lost and Found dataset. This can further be extended to datasets with indoor conditions like factories and warehouses where autonomous vehicles can navigate safely. Also, this work can be extended to perform in different challenging conditions and environment like foggy and rainy weather with poor visibility.

REFERENCES

- [1] J. Long, E. Shelhamer, and T. Darrell, “Fully convolutional networks for semantic segmentation,” in Proceedings of the IEEE conference on computer vision and pattern recognition, 2015, pp. 3431–3440.
- [2] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, “Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs,” IEEE transactions on pattern analysis and machine intelligence, vol. 40, no. 4, pp. 834–848, 2017.
- [3] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, “Encoder Decoder with atrous separable convolution for semantic image segmentation,” in Proceedings of the European conference on computer vision (ECCV), 2018, pp. 801–818.
- [4] L. Deng, M. Yang, T. Li, Y. He, and C. Wang, “Rfbnet: deep multimodal networks with residual fusion blocks for rgb-d semantic segmentation,” arXiv preprint arXiv:1907.00135, 2019.
- [5] W. Wang and U. Neumann, “Depth-aware CNN for RGB-D segmentation,” CoRR, vol. abs/1803.06791, 2018. [Online]. Available: <http://arxiv.org/abs/1803.06791>
- [6] C. Couprie, C. Farabet, L. Najman, and Y. LeCun, “Indoor semantic segmentation using depth information,” arXiv preprint arXiv:1301.3572, 2013.
- [7] S. Ramos, S. Gehrig, P. Pinggera, U. Franke, and C. Rother, “Detecting unexpected obstacles for self-driving cars: Fusing deep learning and geometric modeling,” in 2017 IEEE Intelligent Vehicles Symposium (IV). IEEE, 2017, pp. 1025–1032.

[8] C. Hazirbas, L. Ma, C. Domokos, and D. Cremers, “Fusenet: Incorporating depth into semantic segmentation via fusion-based cnn architecture,” in Asian conference on computer vision. Springer, 2016, pp. 213–228.