FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Synthetic Data in Semantic Segmentation and Lane Estimation for Autonomous Driving

João Filipe Delgado Lemos de Matos

International Master in Computer Vision

Supervisor: João Nuno Duarte Fernandes, PhD

Supervisor: Prof. Jaime dos Santos Cardoso, PhD

February 22, 2023

Synthetic Data in Semantic Segmentation and Lane Estimation for Autonomous Driving

João Filipe Delgado Lemos de Matos

International Master in Computer Vision

Approved in oral examination by the committee:

Chair: Prof. Ana Maria Mendonça

External Examiner: João Tiago Ribeiro Pinto

Supervisor: João Nuno Duarte Fernandes

February 22, 2023

Abstract

Autonomous driving vehicles, an emerging industry, come to provide multiple benefits, including reducing the amount of incidents caused by human error, improving traffic and allowing people with limited mobility to travel independently. Being a recent industry means that the amount of data publicly available is very limited, also for the fact it can be really costly. In these vehicles, the perception module plays a crucial role, being responsible for perceiving the environment and objects around, using various sensors, such as cameras, LiDAR, radar, and ultrasonic. For this reason, the data used for the perception module is essential to develop methods or deep learning models for recognition tasks, for example, object detection, drivable area segmentation, lane segmentation, and many others. The goal of this work is to study the impact of using synthetic data from autonomous driving to train deep learning models, in this case, YOLOP and HybridNets, for semantic segmentation of the drivable area and the lane lines in real-world scenarios. To achieve this, a custom dataset of RGB images was generated using the CARLA simulator, which includes a range of weather conditions, environments and traffic density present in the images. The dataset was then used to train both models. Transfer learning and domain adaptation were then applied to these newly trained models and tested using the BDD100K dataset. Additionally, two ablation studies were conducted to verify how the models would perform with ego-lane distinction on the drivable area task and to compare the performance between the single and multi-head approach.
The results of the study were not able to overcome the state-of-the-art results, with YOLOP achieving 86% mIoU and 24% mIoU, and HybridNets achieving 86% mIoU and 18% mIoU in drivable area segmentation and lane line detection respectively. Nonetheless, a valuable synthetic dataset for autonomous driving was created and has the potential to be a useful resource for future researches or industry applications.

Keywords: Deep learning, Autonomous driving, CARLA, BDD100K, YOLOP, HybridNets, Synthetic

Resumo

Veículos de condução autónoma, uma industria em crescimento, vem para fornecer múltiplos benefícios, incluindo redução de acidentes causados por erro humano, melhorar o tráfego e proporcionar a possibilidade a pessoas com mobilidade reduzida viajarem de forma independente. Sendo uma indústria recente significa que a quatidade de dados disponíveis públicos é bastante limitada, também pelo facto de ser possível tornar-se bastante dispendiosa. Nestes veículos, o módulo da percepção desempenha um papel crucial, sendo responsável por observar o ambiente e os objetos há volta, utilizando vários sensores, como câmaras, LiDAR, radar, e ultrassónico. O objetivo deste trabalho é estudar o impacto de usar dados sintéticos de condução autónoma para treinar modelos de aprendizagem profunda, neste caso a YOLOP e a HybridNets, para a segmentação semântica da área dirigível e as linhas das faixas em cenários do mundo real. Para alcançar isto, foi gerado um dataset sintético personalizado com imagens RGB, usando o simulador CARLA, para treinar ambos os modelos. Este inclui uma variedade de condições climatéricas, ambientes e densidade do tráfego presente nas imagens. De seguida, usando o dataset real BDD100K, aplicou-se as técnicas de transferência de aprendizagem e adaptação de domínio aos modelos recém treinados. Posteriormente foram testados sobre o mesmo dataset. Adicionalmente, dois estudos extras foram realizados de modo a verificar como é que os modelos iriam desempenhar com a distinção de ego-lane na tarefa de área dirigível e comparar o desempenho entre as abordagens de única ou múltiplas cabeças.
Os resultados do estudo não foram capazes de superar os resultados do estado da arte, com YOLOP obtendo 86% mIoU e 24% mIoU, e HybridNets alcançando 86% mIoU e 18% mIoU nas tarefas segmentação de área dirigível e deteção de faixas respetivamente. Contudo, foi criado um dataset sintético para condução autónoma com potencial de vir a ser um recurso útil para futuras investigações ou aplicações na indústria.

Keywords: Aprendizagem profunda, Condução autónoma, CARLA, BDD100K, YOLOP, HybridNets, Sintético

Acknowledgements

Completing this thesis and degree was an incredible challenge, packed with amazing and joyful experiences that I will carry with me forever.
I would like to express my gratitude to both of my supervisors, Prof. Jaime Cardoso and João Fernandes, for their invaluable patience and insightful feedback during the realization of this thesis. Also to the project THEIA for giving me the opportunity to work on this wonderful topic, and for providing their computing resources to train and test the deep learning models.
A special thank you to my cousins Solange and Diogo, my aunt Elda and uncle Luis, for their availability and support during my trips to the north of Portugal and Spain.
Finally, last but not least, a big thank you to my parents, my girlfriend Sofia, and other close family and friends, for their continuous support during the realization of this master's degree.

João Matos

This work was partially supported by European Structural and Investment Funds in the FEDER component, through the Operational Competitiveness and Internationalization Programme (COMPETE 2020) [Project nº 047264; Funding Reference:POCI-01-0247-FEDER-047264]

Contents

List of Figures

List of Tables

Abbreviations

AI	Artificial Intelligence
API	Application Programming Interface
BCE	Binary Cross Entropy
BiFPN	Bidirectional Feature Pyramid Network
CE	Cross Entropy
CLRNet	Cross Layer Refinement Network
DCLGAN	Dual Contrastive Learning Generative Adversarial Networks
DVS	Dynamic Vision Sensor
FL	Focal Loss
FPN	Feature Pyramid Network
FPS	Frame per Second
GAN	Generative Adversarial Network
GNSS	Global Navigation Satellite System
GPU	Graphics Processing Unit
GTA5	Grand Theft Auto V
HDRP	High Definition Render Pipeline
HSV	Hue Saturation Value
IMU	Inertial Measurement Unit
IoU	Intersection over Union
JSON	JavaScript Object Notation
K-Lane	KAIST-Lane
LiDAR	Light Detection and Ranging
LLDN-GFC	LiDAR Lane Detection Networks utilizing Global Feature Correlator
LKAS	Lane Keeping Assist System
NPC	Non-Playable Character or Non-Player Character
NRE	Neural Reconstruction Engine
PAN	Path Aggregation Network
POV	Point of View
RGB	Red Green Blue
ROI	Region of Interest
SPP	Spatial Pyramid Pooling
UDA	Unsupervised Domain Adaptation
UE4	Unreal Engine 4
YOLOP	You Only Look Once for Panoptic Driving Perception

Chapter 1

Introduction

1.1 Background

Autonomous driving is a cutting-edge technology that has been gaining significant traction in both research and industry consumer purposes. The ability of vehicles like this to operate autonomously relies on the interaction of multiple modules, each one designed to be responsible for a specific task. For example, the decision making module, where receives continuously information from the outside world and makes decisions based on that data. Another example is the sensory module, which is responsible for using a variety of sensors to extract information from the outside environment, these sensors could be LiDAR, radar, camera-based, etc. It is crucial for autonomous vehicles to have a comprehensive understanding of their surroundings. This includes knowing their location, detecting other objects present, such as vehicles, traffic signs and pedestrians, and also detecting the road layout, where it is possible to drive and segment each lane line separately. Being an emerging industry, autonomous driving faces a problem of data availability, meaning that there are not a lot of publicly available datasets that can be used to train and test these systems. Gathering data for this kind of tasks can not only pose a problem because of the complexity and danger associated with it, as it involves driving a car in the public but also the high cost of acquiring the necessary material, such as the sensor, the vehicle and the material to attach and prepare the sensors. Due to these problems, researchers and companies look to invest in the development of simulation environments, where they can safely put together the amount of data desired, replicate improbable or dangerous scenarios that could happen in the real world and alter the environments settings as desired. This allows, saving a lot of time and money without taking any risks.

1.2 Context and Goals

The goal of this thesis is to research the impact of using synthetic generated data from autonomous driving simulators on the deep learning models, such as YOLOP [52] and HybridNets [50], for the

tasks of detecting and segmenting the drivable area and the lane lines present in the road, and evaluate the ability of these models to generalize from the synthetic to real-world scenarios. To achieve this, a synthetic dataset of RGB images was constructed using CARLA [13] simulator. CARLA is more than capable for this task, due to the wide range of environment's, weather combinations, traffic management and sensors available. After both models are trained with the synthetic dataset, is performed transfer learning and domain adaptation using the BDD100K [54]. The models are then evaluated with the same BDD100K dataset. Finally, comparative studies were conducted to examine the model's performance with the ego-lane distinction in the drivable area task, and the difference between single a multi-head approach.

1.3 Structure

The thesis structure begins with Chapter 2 which provides an introduction to the theory about image segmentation, drivable area, lane lines segmentation and also some metrics used to evaluate the model's performance. The Chapter 3 presents a literature review on state-of-the-art technologies, including detection algorithms, deep learning models and autonomous driving simulators. In Chapter 4 is detailed the synthetic dataset created for this work, specifying all the details that were taken into account, such as the amount of data acquired, how the data acquisition was made, how the data was split, etc. Chapter 5 explains all the methodologies that were done to perform the training with the synthetic, data and the transfer learning and domain adaptation to the real world. The Chapter 6 presents the obtained results throughout all experiments and where the main conclusions took place. The final Chapter 7 mentions future work suggestions that could be done as further research on this topic.

Chapter 2

Fundamentals

2.1 Image Segmentation

Image segmentation is the process of dividing the image into multiple segments, also known as image regions. The objective of this task is to segment multiple regions that have different semantic meanings. The algorithms for segmenting monochromatic images are generally divided into two different categories, based on their approach to deal with properties of intensity values: discontinuity-based and similarity-based [16].

The first category assumes that the boundaries of the regions are distinct enough from each other and the background to allow boundary detection based on local intensity discontinuities. The primary approach used in this category is edge-based segmentation, which utilizes methods such as Canny-Edge [9] or the Sobel Operator [43] to detect the drastic changes in intensity values, known as edges, and then performs edge linking techniques to combine the adjacent edges and form a whole object.

The latter category is based on the idea of dividing the image into smaller regions based on predefined criteria. This category uses region-based segmentation approaches, in which regions are grown by recursively connecting neighboring pixels that have similar properties. Such techniques include K-means Clustering [12][56] and Region Growing [4].

The advancement in computational power has led to the increased use of deep learning models and their applications. Early versions of Convolutional Neural Network was primarily used for image classification, but soon after with the development of the Fully Convolutional Neural Network [41], it became possible to classify each pixel with its corresponding class, instead of classifying the whole image with a single class. This enables the use of these models to perform object detection, semantic segmentation and image classification, the latter being possible if the output is a single value instead of a whole image.

2.1.1 Drivable Area Segmentation

The segmentation of drivable area is a crucial task for the perception module of an autonomous vehicle, being able to not only know where the road is but also which part of the road the car can drive. This task can be divided into two categories, first one is simply to detect and segment which side of the road the car can drive and check if any car is present, example in Fig. 2.1, the other option is basically the same thing but adding the ability to distinguish which lane the car is driving, if exists any adjacent lanes and determine if are drivable for the vehicle, as seen in Fig. 2.2.

Figure 2.1: Drivable area example without ego-lane. [52]

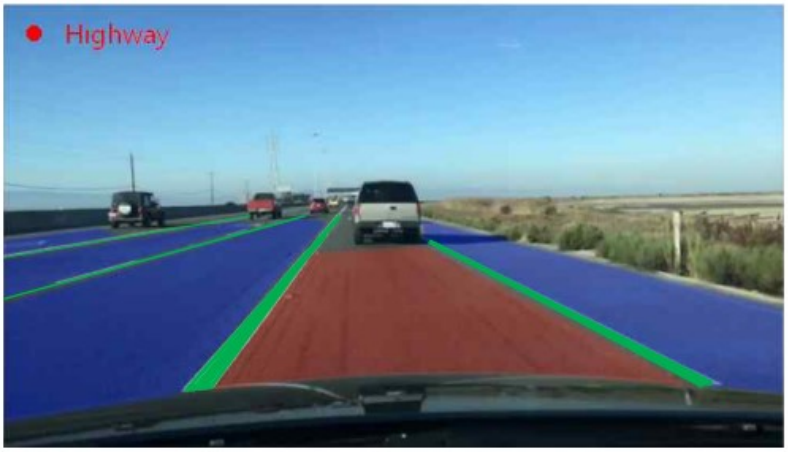


Figure 2.2: Drivable area example with ego-lane. [26]

2.1.2 Lane Segmentation

The goal of lane segmentation is to accurately identify the boundaries of each lane on the road, so that the vehicle can perceive properly each lane individually and its own position relatively to the

lanes, as shown in the Fig. 2.3. This information is used later by the vehicle's perception module to help the car make decisions.

Figure 2.3: Lane lines example without ego-lane. [52]

2.2 Metrics

Despite a wide range of available metrics in the task of image segmentation in the state of the art, the following were used:

- Pixel Accuracy
- Intersection over Union (IoU)
- F1-Score

2.2.1 Pixel Accuracy

The pixel accuracy metric is expressed as the ratio of the number of pixels in the output image that has the same value as the target label, by the total number of pixels in the image, demonstrated in the Equation 2.1.

$$PixelAccuracy(output, target) = \frac{\sum output == target}{H \times W} \tag{2.1}$$

2.2.2 Intersection over Union

The intersection over union, also known as Jaccard index, is one of the most used metrics for image segmentation [48] and it is extremely effective to verify if the model is performing correctly its task.

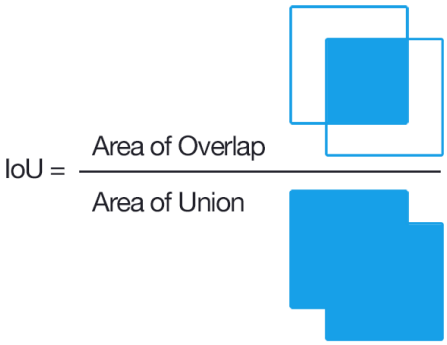


Figure 2.4: IoU metric. [3]

$$JaccardIndex = \frac{TP}{TP+FP+FN} \tag{2.2}$$

The IoU metric, represented in Fig. 2.4, measures the overlap area between the output from the models and the target labels, divided by their union. This is mathematically demonstrated by the Equation 2.2 and an example of this metric is illustrated in Fig. 2.5.

Figure 2.5: Demonstration of the IoU metric. Green rectangle is the ground-truth and the red one is the predicted. [3]

2.2.3 F1-Score

The F1-Score, also known as dice coefficient, is relatively similar to the IoU metric, as can be noted in Equation 2.3.

$$F1-Score = \frac{2 \times TP}{2 \times TP+FP+FN} = 2 \times \frac{Precision \times Recall}{Precision+Recall} \tag{2.3}$$

This metric is basically the harmonic mean of precision and recall, meaning computes the average between them and giving the same weight.

2.3 Loss Functions

The loss functions have the purpose of providing feedback during the training of deep learning models, and to guide them to approximate the ideal solution which maps the input to the desired output data.
For image segmentation, the most commonly known losses are:

- Cross-Entropy Loss or Binary Cross-Entropy Loss
- Intersection over Union Loss
- Dice Loss
- Focal Loss

2.3.1 Cross-Entropy Loss or Binary Cross-Entropy Loss

The cross entropy loss measures the uncertainty between the predicted probability and the class distribution, see Equation 2.4. In case of image segmentation is used to compare the predicted probability (p) of each pixel belonging to the true class (t) out of all C classes, the binary cross-entropy is the same process but only between 0 and 1 labels, expressed in the Equation 2.5.

$$L_{CE} = -\sum_{i=1}^{C} t_i \times log(p_i) \tag{2.4}$$

$$L_{BCE} = -[t \times log(p) + (1-t) \times log \times (1-p)] \tag{2.5}$$

2.3.2 Intersection over Union Loss

The Intersection over Union, as mentioned earlier, one of the most used metrics has its own loss function to help the models during training, noted in the Equation 2.6.

$$L_{IoU} = 1 - \frac{TP}{TP+FP+FN} \tag{2.6}$$

2.3.3 Dice Loss

Similarly to the IoU, the F1-Score metric has its own loss representation, that is widely used for medical image segmentation tasks, mathematically represented in the Equation 2.7.

$$L_{Dice} = 1 - \frac{2 \times TP}{2 \times TP+FP+FN} \tag{2.7}$$

2.3.4 Focal Loss

The Focal Loss [28] was originally designed for object detection tasks, and to deal with class imbalance problems, represented in the Equation 2.8. In case of the image segmentation, class

imbalance happens when the pixels of a desired label is much lower in quantity compared with the background pixels.

$$L_{FL} = -\alpha_t \times (1 - p_t)^{\gamma} \times log(p_t) \tag{2.8}$$

Chapter 3

State of the art

This segment provides a review of relevant contributions among the topic of drivable area and/or lane segmentation. Initially, it will be described a few traditional computer vision techniques, followed by state-of-the-art deep learning methods, including different architectures and functionalities.

3.1 Traditional Computer Vision Methods

3.1.1 Lane Marker Estimation Method

This technique [37], an improvement from the previous algorithm [36] presented by the same authors, shown in Fig. 3.1, guarantees multi-lane detection while also detecting the lane markers. The detection of lane markers is crucial information in order to acquire longitudinal measurements of the road and its potential lanes. This method is based on a confidence map, a grayscale image where the intensity of the pixels corresponds to the confidence being a road marking pixel, as seen in Fig. 3.1. It is generated with different types of high-intensity brightness transitions detectors and a Gaussian filter.

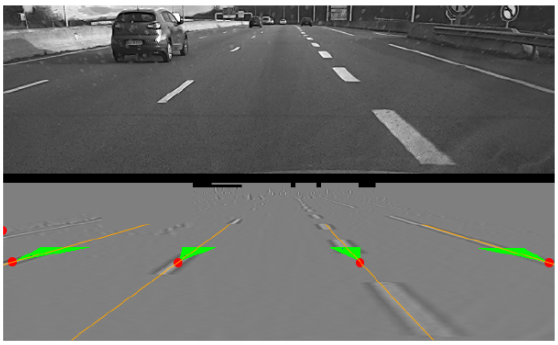


Figure 3.1: Previous lane detection algorithm. Top image: original image. Bottom image: Lane detection. Red circle: Lane agent. Green triangle: perception triangle of agents. Orange line: road marking estimation modeled by a smoothing cubic spline. Background: confidence map. [36]

After the confidence map being built, in order to detect the lane markers, it is used a gray level profile p that is described as a function of distance along the estimation over the gray level in the map [37]. Then to filter out the unwanted noise that can be generated by some small objects present on the road, the profile is convoluted with a Normal distribution $\mathcal{N}(\mu, \sigma^2)$, where the standard deviation σ corresponds to the maximum observable length in the road and μ is the abscissa of the given point in the profile interpolation, shown in Fig. 3.2. The Gaussian profile is then thresholded by the parameter gt, representing the minimum gray level for a lane marker estimation.

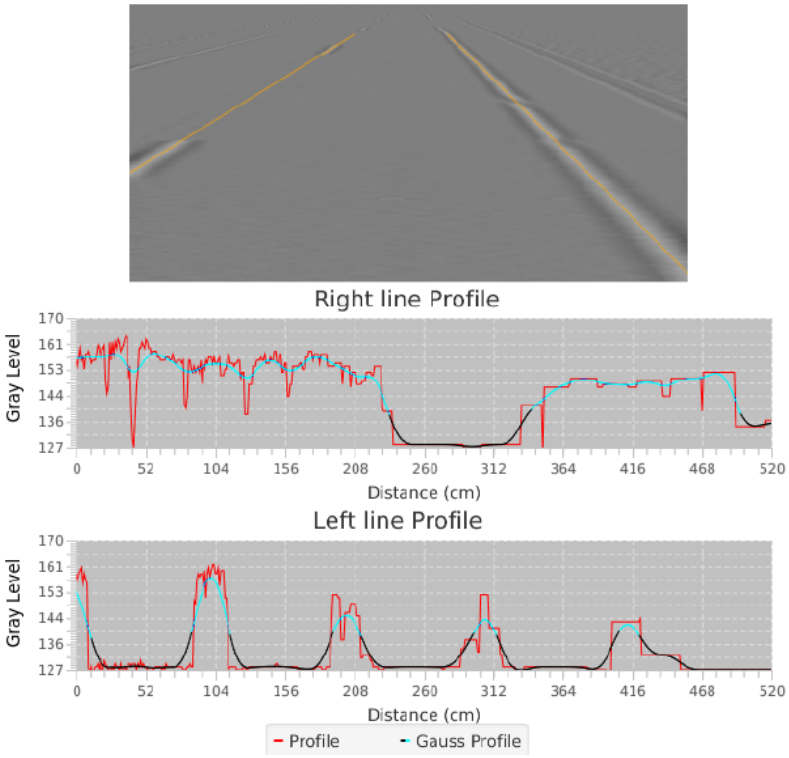


Figure 3.2: Profile: gray level profile of lane estimation. Gauss profile: profile convoluted with the Normal distribution. Blue parts: lane marker area estimation. Black parts: no information area. [37]

This initial approach has its own problems for only being able to detect some parts of the lane marks, as can be seen in the left line of the Fig. 3.2. Therefore, the proposed approach to improve this detection, was to convolve the Gaussian profile with a unit step function kernel H, as illustrated in Fig. 3.3, in the interval A_r and an inverted kernel in the interval A_l. These intervals represent the rising and the decay of the lane marker edges, $A_r = [L_a, M_a]$ where L_a and M_a are the upstream and downstream knee-point for a given rising edge, and $A_l = [M_a, U_a]$ where M_a and U_a are the downstream and upstream knee-point for a given decay edge, therefore the marker area $A = \bigcup[A_r, A_l]$.

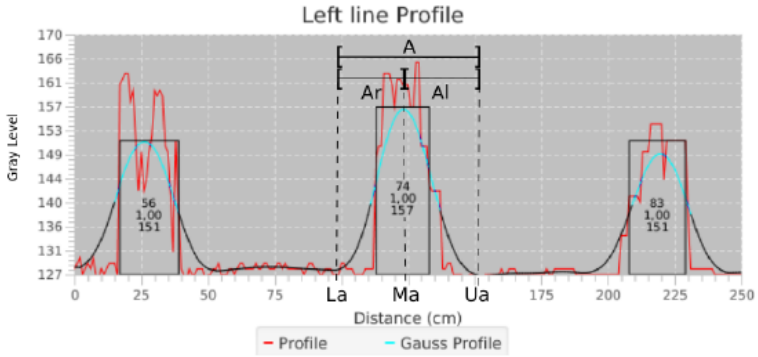


Figure 3.3: Profile: gray level profile of lane estimation. Gauss profile: profile convoluted with the Normal distribution. Black box: lane marker geometry estimation with id and median estimation (Top to bottom). [37]

The tracking is based on the Transferable Belief Model framework, which compares all possible associations with the use of basic belief assignments. This belief criterion was calculated from the similarity between detected and estimated objects, by modeling the noise that can disrupt the position of the lane tracker along the lane estimation. The criterion was calculated under two of the most disturbing factors:

- The pitch error between two iterations, due to vehicle acceleration or braking which causes an X-axis distortion.
- The error on the assumption of the planar world, which is the function of the distance between the lane marker and the ego vehicle.

Ultimately, this approach proves to be much more robust than its predecessor, achieving a detection rate over 95% accuracy for ego lane markers and 93.5% for multi-lane markers, as demonstrated by the number of correctly detected lane markers. An example of this detection can be seen in Fig. 3.4.

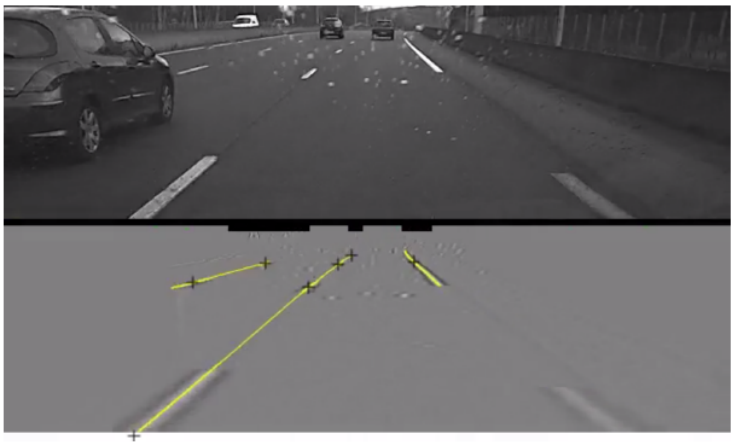


Figure 3.4: Example lane marker detection with water drop on the windshield. [37]

3.1.2 Road Detection using Color Classification

A prevalent approach [5] for road detection is to utilize color features to classify pixels. These algorithms reduce the effect of changes in lighting and weather by making use of the discriminatory/invariant properties of different color representations.
This is an online method that contains two different stages: color conversion and pixel classification. In essence, first transform the RGB image to a preferred color representation that is better suited for the problem in hand, and then serves as input to the single class classifier that will output a class for each pixel, see Fig. 3.5.

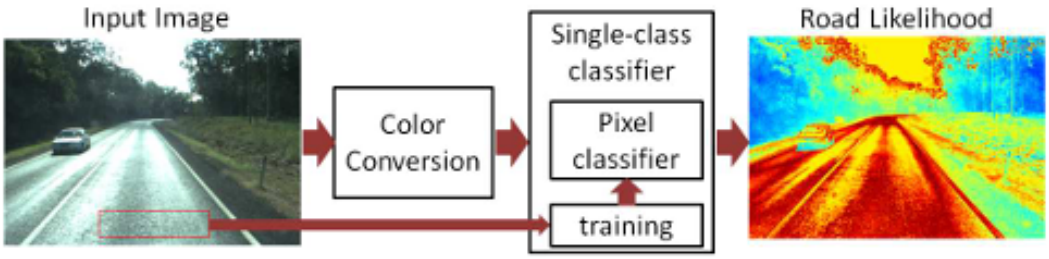


Figure 3.5: Online road detection algorithm. [5]

During the development of the algorithm, research was conducted to evaluate the various color spaces and their sensitivity to different environmental conditions. The chosen color spaces were as follows:

- RGB
- normalized RGB
- opponent color space
- HSV
- CIE-Lab

Each of these spaces has different color properties that are displayed in the Table 3.1. In summary, the color channels R, G and B have great discriminative features but lack invariance to shadows and lighting. In contrast, the hue and saturation provide more invariance however low discriminative power.

Similarly to the pixel conversion, another study was done about the performance of different binary classifiers. These can be divided into three different categories, density based, reconstruction based or boundary based. Density classifiers focus on modeling the probability density function of the target class in the training data, meanwhile, the reconstruction classifiers are based on the assumption of the structure of the data, and the boundary classifiers search for the boundaries

Table 3.1: Properties of different color representations from a theoretical point of view. Robustness to each property is indicated by '+' and weakness by '-'. [5]

	R	G	B	nr	ng	O1	O2	L, V	ab	H	S
Global illumination changes	-	-	-	-	-	+	+	-	+	+	+
Discriminative power	+	+	-	-	+	-	-	-	+	-	-
Distance to the camera	+	+	+	+	+	+	+	+	+	+	+
Strong shadows	-	+	-	-	-	+	+	-	+	+	+
Highlights	-	-	-	-	-	+	+	-	+	+	+
Road type and shape	+	+	+	+	+	+	+	+	+	+	+

that surround all the elements from the target class in the training set. The classifiers tested were the following:

- **Model-based (Mb)**: This is a non-parametric classifier that uses a likelihood measure to approximate the conditional probability of having a road pixel given a pixel value. [5]
- **Single Gaussian (G)**: This classifier models road training samples using a unique Gaussian distribution. [5]
- **Robustified Gaussian (RG)**: The single Gaussian classifier is sensitive to outliers and noise in the training samples. [5]
- **Mixture of Gaussians (MoG)**: Single Gaussian classifiers have the drawback of modeling a single distribution. [5]
- **k-means (km)**: This classifier does not rely on estimating the density probability function. [5]
- **k-center (kc)**: This method aims at covering the training set with k small balls with equal radius. [5]
- **Principal Component Analysis (PCA)**: This classifier describes road data using a linear subspace defined by the eigenvectors of the data covariance matrix. [5]
- **Nearest neighbor (NN)**: This method avoids the explicit density estimation and estimates the road likelihood using the distances between test pixels and the training data. [5]
- **Linear programming distance-data description (dLP)**: This method aims at describing the road data in terms of distances to other objects. [5]
- **Support Vector Descriptor (SVD)**: This method aims to define the hypersphere with a minimum volume covering the entire training set. [5]
- **Minimax Probability (MPM)**: This method aims at computing the linear classifier that separates the data from the origin rejecting maximally a specific fraction of the training data represented as a random variable. [5]

- **Minimum Spanning Tree (MST)**: This is a non parametric classifier aiming at capturing the underlying structure of the data based on fitting a minimum spanning tree to the training data. [5]

In terms of results, the combination of the Robustified Gaussian classifier and HS color space was the one that achieved the best results with a 93.4% accuracy. Another benefit of this algorithm is its simplicity, which makes it capable of performing these uncomplicated tasks in real time.

3.1.3 Illumination-Robust Road Detection Approach

Illumination changes are a factor that continuously happens during the driving experience of a car. It can occur from the frequency of lightning poles present illuminating the road, to the shadows originated from external objects. This novel approach [53] tackles these problems by taking the grayscale image with modified saturation, detecting the road boundary lines, and then with the help of a feature-based method identify lane markings.
Initially, the image goes through a preprocessing stage where an ROI is extracted, in order to remove irrelative information. Typically the road information is concentrated on the bottom half of the image, so it is extracted the lower two-thirds of the image as the ROI. Then a color conversion to a grayscale image needs to take place to lower the computational complexity afterwards, but a problem still persists that the RGB color space is sensitive to illumination changes. Therefore, there is a need for a transformation to another illumination-insensitive color space. The proposed approach was to modify the saturation from the HSV color space, see Equation. 3.1, to diminish string shadows that may be present on the road, see Fig. 3.6.

$$S' = \frac{max(R,G,B) - B}{max(R,G,B)} \tag{3.1}$$

(a) (b) (c)

Figure 3.6: (a) RGB image of road with strong shadows. (b) S component grayscale image. (c) S' component grayscale image. [53]

The following stage of road boundary detection starts with the image segmentation, where it is performed three successive operations of thresholding, morphological filtering and connected component analysis. Respectively to convert into a binary image, remove noise, and remove small isolated areas, for example Fig. 3.7.

(a) (b) (c)

Figure 3.7: (a) Binaryzation. (b) Morphological filtering. (c) Small isolated regions removed. [53]

At this instant, the boundary points should be ready to be detected, the proposed approach begins with a bottom-up scan of each column of the image until encounters the first non-zero (boundary) pixel. These pixels are candidates for the boundary points. Then the same search is applied but to each row of the candidate points from the middle to the left side of the image, then another search from the middle to the right side of the image, as shown in Fig. 3.8.

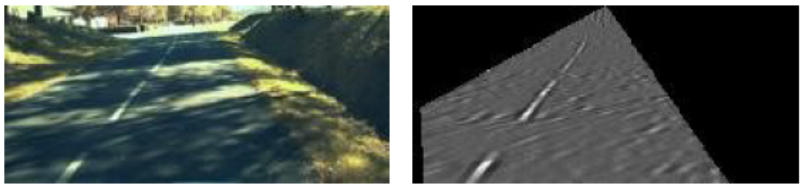
(a) (b) (c)

Figure 3.8: (a) Candidate points. (b) Left boundary points after a middle-left scan. (c) Right boundary points after a middle-right scan. [53]

After detecting all the boundary points of the road, the Hough Transform is used to extract the best-fitting straight lines.
Finally, the last stage corresponding to the lane marking detection starts with the ROI adjustment by removing the region above the horizon to reduce further computational time. Subsequently, to extract features that are insensitive to lighting and shadow variations, the authors propose a formula (as outlined in Equation. 3.2). The formula detects lane markings by identifying pixels with a high-intensity difference compared to their surrounding pixels, as shown in example Fig. 3.9.

$$V'(u,v) = \frac{V(u,v)}{V(u,v-w_u)+V(u,v+w_u)} \tag{3.2}$$

Where $V(u,v)$ and $V'(u,v)$ represent the pixel and the computed feature at row u, column v, and w_i represent the pixel width of the lane marking in the i-th row.



(a) (b)

Figure 3.9: (a) ROI adjustment result. (b) Lane marking feature extraction result using Equation 3.2. [53]

Likewise the previous stage, the lane-markings features image will be performed a simple fixed-thresholded and then applied the Hough transform to detect all the lines, shown in Fig. 3.10.

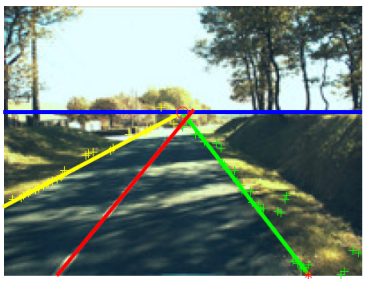


Figure 3.10: Detection result. The left (yellow) and right (green) boundaries are extracted by fitting the boundary points (green cross). The intersection of those lines is defined as vanishing point (red circle). The lane marking (red) is extracted from image below the horizontal line (blue). [53]

Even though this is a robust approach, it still experiences some flaws, e.g. large curvature roads may lead to some miss detections with the Hough transform. The shadow-diminishing in preprocessing stage should be improved to an adaptive alternative, and also the ROI extraction needs to be adaptive depending on the angle and position the camera is placed in relation to the road.

3.2 Deep Learning Methods

Unlike the traditional methods, most of the deep learning models follow the same architectural principles. The ability to perform real-time inference is important for an autonomous driving application. Therefore, the models need to work fast and accurately for drivable and/or lane line segmentation.

The overall architecture of the models is built upon encoder-decoder architecture, where the encoder is shared by all tasks and the features are then divided into multiple heads, one for each task. Through this way, performing multiple tasks simultaneously from the shared computation of the features, saving computational time.

The backbone, an essential part of the model, responsible to extract and organizing features from the input images. Most of the cases reuse pre-trained networks, that attained high accuracy and efficient performance, e.g. using the ImageNet dataset. For example, in case of YOLOP [52], adapted for panoptic driving perception, it follows the same principle of YOLOv4 [7] using the CSP-Darknet53 [51] as the backbone. Due to its outstanding performance and capabilities to solve gradient duplication problems during optimization, this method also supports feature propagation and feature reuse, resulting in a decrease in required parameters and computations. For instance, the HybridNets [50] model makes use of EfficientNet-B3 [46] as the backbone that also solves the issue of network optimization by searching parameters based on the neural architecture search to design a stable network. In case of DLT-Net [35], it takes advantage of the VGG16 [42] and the feature pyramid structure [34] to encode deep a multi-sized features.

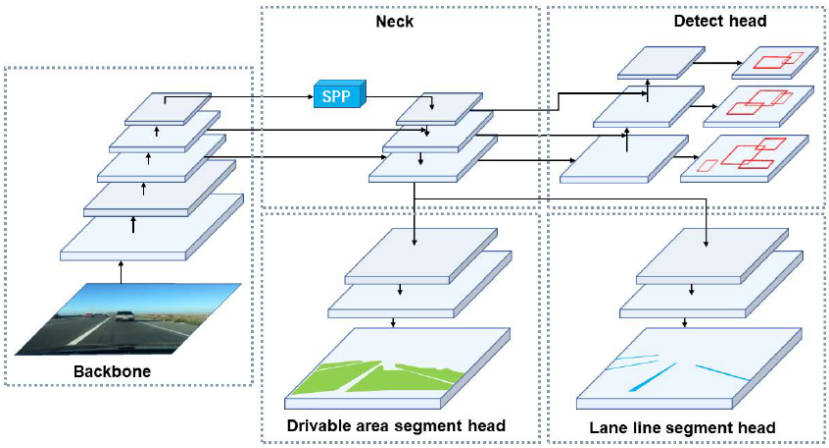


Figure 3.11: YOLOP model architecture. [52]

Following the backbone, each model start to differ from each other, since up to this moment, it concerned just to extract and build the features from the input image. After this, with the features arranged some models take an extra stage called neck, as seen in the Figs. 3.11 and 3.12 corresponding to the YOLOP and HybridNet architectures. The neck stage is responsible to fuse the features together generated by the backbone. In case of YOLOP, it forms a combination of both SPP [20] and a FPN [27] to be able to extract and generate features at different scales and different semantic levels. For the HybridNets model, it has a BiFPN network based on the EfficientDet [47], basically a bidirectional FPN that allows to fuse features at different resolutions on a cross-scale connection giving therefore the possibility to learn the importance of each feature for each level by adding weight. The DLT-Net does not have a neck, because it already fuses the features at the end of the encoder by combining neighborhood layers by using the mean operation and thus combining deep features with shallow features.

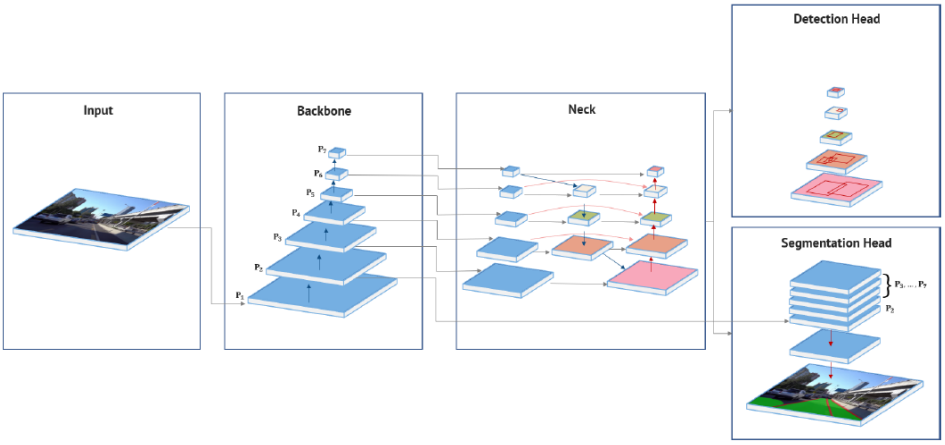


Figure 3.12: HybridNets model architecture. [50]

At this point, the features are now ready to be inserted into each of the heads of the model, where the decoder is present and prepared to decode the features in order to output the result of the task assigned.

Concerning YOLOP, it presents three different heads, one for object detection, other for drivable area segmentation, and another for line segmentation tasks. For the object detection head, it uses a PAN, a bottom-up feature pyramid network that receives directly semantic features from multiple levels of FPN fusing them together forming multi-scale feature maps. Both of the segmentation heads present the same process, the bottom layer of the FPN is fed into the branches and after three up-sampling operations, the feature map is restored to its original size (W,H,2).

The HybridNets network possess only two heads, one for detection and another for segmentation. In this case, it performs the drivable area and lane line segmentation together. The detection head receives the multi-scale fusion maps directly from the neck and tries to detect objects without performing any other computation on the features themselves. The segmentation head keeps the features levels P[3] to P[7] from the neck, up-samples each one to have a feature map with size $(\frac{W}{4}, \frac{H}{4}, 64)$. Following the P[2] level from the backbone, it will go through a convolutional layer to have the same feature map size as the previous ones. Then, it fuses all features together and restores the output feature to the size (W, H, 3). The similarity between YOLOP and DLT-NET relies also on the fact that it has also three heads, as seen in Fig. 3.13, one for traffic object detection, and other two for drivable area and lane line segmentation. Each of these heads will apply a simple operation to the final feature map built from the encoder and change it to the size of $(\frac{W}{16}, \frac{H}{16}, 128)$. The drivable area branch simply outputs a feature map with the original size of the input using typical operations of up-sampling. Nonetheless, the novelty of this branch is the fact that the initial feature map is shared with the other branch, called a *Context Tensor*. The reason is the lane lines and the traffic objects are most of the time present in the drivable area. Therefore, the branch by learning to segment this region will help the other heads on their task by sharing the feature map. In the lane line decoder after fusing the tensors together, it will perform the same kind of operations as the previous head, and output a feature map with the size of the original image

and two channels. The object detection head is divided into two parts after the context tensor, one for classification and the other one for bounding box regression.

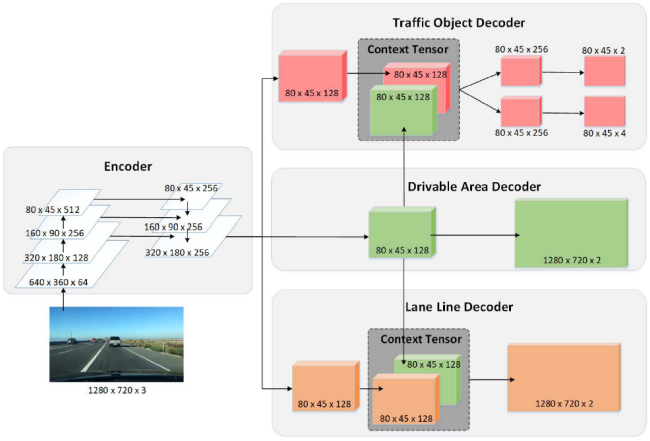


Figure 3.13: DLT-Net model architecture. [35]

In terms of results, the Table 3.2 show the performance that each model acquired from the original papers [35][50][52], all models were tested with the BDD100k dataset [54]. The YOLOP and HybridNets were the ones that were able to achieve better results. Notably, HybridNets achieving significantly a better accuracy on the lane detection task, indicating this approach is the most effective one. On the other side, the DLT-Net, with the strategy of sharing the features from the drivable area branch, did not turn out successfully regarding to the results that were obtained in general.

Table 3.2: Results of the model with the BDD100k dataset. [35][50][52]

	Object Detection (mAP50 %)	Drivable mIoU (%)	Lane Line Accuracy(%)
YOLOP	76.5	**91.5**	70.5
HybridNets	**77.3**	90.5	**85.4**
DLT-Net	68.4	71.3	(missing)

Looking at the research done by others on different datasets, Zheng et al. [55] developed a network called Cross Layer Refinement Network (CLRNet), which combines high and low-level features from the lanes. This network starts detecting the high-level semantic features and then refines with the low-level features, to extract more contextual information and local detailed features. The authors then tested the network on three different public datasets, TuSimple [1], CULane [33] and LLAMAS [6], achieving an F1-Score of 97.89%, 80.47% and 97.16% respectively.
Although, this thesis focuses on RGB camera-based data, it is important to note other methodologies that use light detection and ranging (LiDAR). For instance, Paek et al. [31] propose KAIST-Lane (K-Lane), the first and largest public lane dataset using LiDAR. They also presented LiDAR

lane detection networks utilizing a global feature correlator (LLDN-GFC), capable of exploiting spatial characteristics of the lane lines in the cloud, achieving an amazing F1-score of 82.1%.

3.3 Autonomous Driving Simulators

Driving simulators offer the possibility to train and test models for autonomous driving research. Simulators have a high flexibility associated with them, meaning that can generate different kinds of scenarios that would be difficult to recreate in the real world, such as dangerous situations of cars colliding with each other, pedestrians or cyclists running in front of the cars, vehicles going off the road and many others. An additional possible solution involves using video games, e.g. Grand Theft Auto V, commonly known as GTA5. GTA5 is a highly realistic and detailed open-world video game. It offers a virtual world with a variety of environments, weather conditions, traffic, and non-player characters (NPCs) with realistic behaviors. This realism and attention to detail make GTA5 an ideal candidate as a simulator for autonomous driving research. The game can be used to extract real-world scenarios and generate a synthetic dataset for training autonomous driving models. On the other hand, usually they do not support any kind of customization of the actors in the world. They have limited diversity of usable sensors, no fidelity on the driving policies incorporated into the actors or any event feedback that may happen to the agent.
CARLA [13] is an open-source simulator that was developed from scratch. It includes urban layouts, a multitude of vehicle models, buildings, pedestrians, street signs, etc. This simulator was built upon the open-source game engine UE4 [14], a powerful software capable of state-of-the-art rendering quality, realistic physics, NPC logic and an ecosystem of plugins available. CARLA is designed as a server-client architecture, where the server is responsible to host and simulate the world and NPCs. The client API, implemented in Python, is in charge of the interaction between the autonomous agents and the server by sending commands and meta-commands. There are commands that control the vehicle itself, and meta-commands that control the world, which can be used to reset the world and its agents, change properties of the environment such as the map, weather, sensors used, illumination and density of the cars and pedestrians. CARLA has a variety of sensors, like detectors of collisions, lane invasions, obstacles, or even for navigation and measuring purposes for example GNSS, IMU, LiDAR, etc. CARLA also provides multiple semantic classes: road, lane-marking, traffic sign, sidewalk, fence, pole, wall, building, vegetation, vehicle, pedestrian, and others. As can be seen in the right image of Fig. 3.14.
However taking into account the focus/scope of this thesis, a more detailed analysis will be given to image sensor data. Currently, CARLA has the following camera sensors:

- Depth
- RGB
- Optical Flow
- Semantic Segmentation

- Instance Segmentation
- DVS

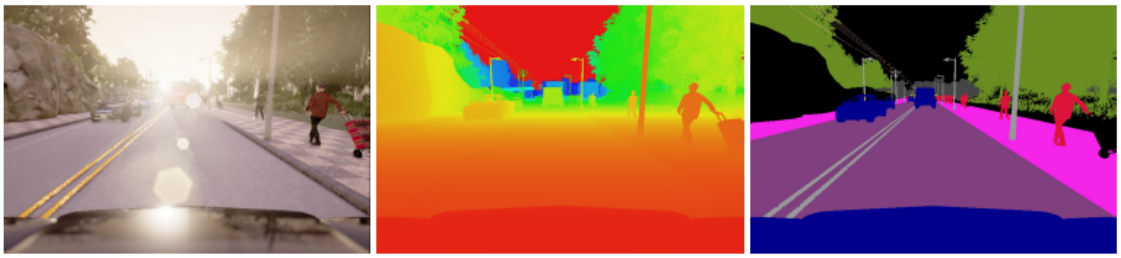


Figure 3.14: Three of the sensing modalities provided by CARLA. From left to right: normal vision camera, ground-truth depth, and ground-truth semantic segmentation. [13]

Besides CARLA, there are other open-source simulators that may be worth exploring or investigating. For example, SUMMIT [8] focuses on high-fidelity dense urban traffic on complex realistic world maps, not in terms of graphics but also in world design. SUMMIT takes real-world scenarios of streets and roads as a strategy to provide realistic complex environments. The implementation of the simulator was based on CARLA, to take advantage of the high-fidelity physics, rendering, and sensor, and this way avoiding developing from scratch. Through the python-based API, it can access all the sensor data, semantic information about the classes and objects, meta information about events, vehicle control and planning, etc. The real-world maps are acquired from the software OpenStreetMap [44] and are composed of two topological graphs, a lane network for the vehicles, and a sidewalk for the pedestrians. Then those contexts are taken into account for the behavior model called Context-GAMMA, and guide the traffic accordingly, as it can be observed in Fig. 3.15.

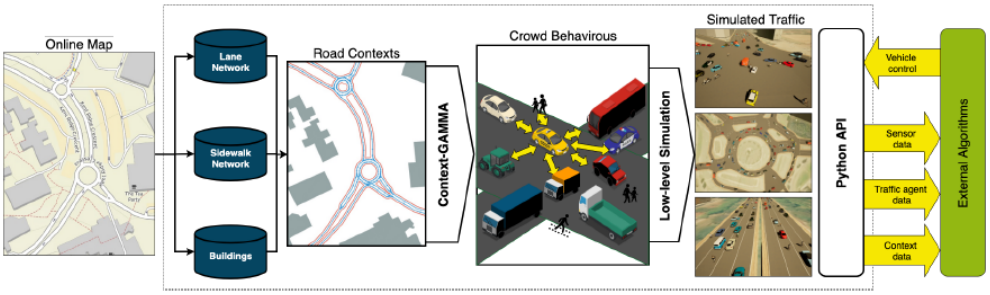


Figure 3.15: SUMMIT simulator overview. [8]

This context-aware behavior model is capable of generating complex interactive behavior of traffic agents. It extends from GAMMA [29], a model that enables real-time simulation and planning for autonomous driving, to include road contexts and use them as constraints, and also the realism and accuracy too. The proposed and extended model provides a way to incorporate road

contexts as objectives and constraints to the velocity space, additionally model traffic rules with help of contextual constraints.
A last simulator example is the LGSVL [39] high-fidelity simulator for autonomous driving, capable of end-to-end full stack simulation, made with the game engine Unity [49] and its strong technology HDRP, capable to provide photo-realistic virtual scenarios that can match the real world. The abilities of the simulation engine can be broken down into four major components:

- Environment simulation
- Sensor Simulation
- Vehicle dynamics
- Simulation of an ego vehicle

In case of the environment simulation module incorporates several functionalities, for example, traffic simulation, weather and time-of-day control. These are crucial features of a simulator being able to modify and adjust to a specific condition, in order to minimize the gap to the real world. The LGSVL simulator includes a web user interface that accepts sensor configurations in JSON files, this facilitates the sensor handling customization. From intrinsic and extrinsic parameters to different types of sensors, positions, publishing rates and also some specific properties, can be all customized in the web user interface.

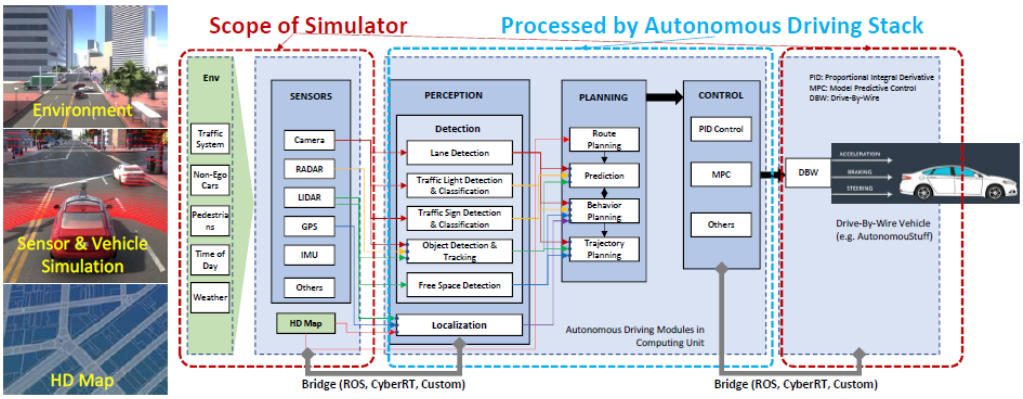


Figure 3.16: LGSVL simulator architecture overview. [39]

In terms of the type of camera sensor images that can provide are the following: RGB color image, depth grayscale image, and semantic and instance segmentation image. Similar to CARLA, there are other types of sensors available to be used, some examples in Fig. 3.16. This simulator offers some further customization, in case of the segmentation images, allowing to configure and selection of which classes are enabled and those who are not, and therefore resulting in an image with the chosen classes segmentation color differentiated from the rest, therefore, preventing further processing to isolate and separate the desired classes.

Table 3.3: Characteristics of the simulators.

	Real World Scenarios	Photo-Realistic	Environment Adjustable	Traffic Control and Customizable	Free
CARLA		X	X	X	X
SUMMIT	X		X	X	X
LGSVL		X	X	X	X

The key attributes of a simulator, as outlined in Table 3.3, include its photo-realism, with models and textures that closely resemble reality, making it easier to perform domain adaptation to the real world. Additionally, the ability to adjust the environment, control the weather and map, and customize traffic, including the number of vehicles, their placement, and types, is the key formula to create scenarios that accurately mimic the real world.
In case of the SUMMIT simulator, the argument presented is that the importance of being able to create high-fidelity scenarios that are similar to the real world, is only valid in specific roads or road segments that are out of the ordinary, because for the most general purposes the majority of the driving is made in a generic and typical road, so this feature is important but not a fundamental one.
Even though explained in detail the last three simulators, there are still many other simulators available, each with its own unique characteristics and features. For instance, the NVIDIA DRIVE Sim, an open-source simulator platform developed by NVIDIA, is currently in a closed early access stage, with high-fidelity 3D environments and multiple sensors available, demonstrated in Fig. 3.17. The unique feature of this simulator is that it provides an AI tool called the neural reconstruction engine (NRE), which is capable of bringing real-world data into the simulation world, and later be customizable using data captured in test vehicles on the roads. This tool is incredibly powerful as it allows to recreate real-world scenarios and still be able to adapt or change anything present to create specific scenarios.

Another simulator is the Cognata, which is versatile and works in multiple industries, including autonomous vehicles where it can be used to train and test autonomous systems. Another area of application is agriculture, where it is designed to train and test perception and control systems of terrains. Additionally, it offers other areas of use such as warehouse, defense, construction, and mining industries.
A few more examples that are worth noting, the AirSim [40] an open-source simulation platform, made by Microsoft Research, developed to research autonomous cars and aerial vehicles like drones. The rFpro originally began as a project of a Formula 1 team capable to test the vehicle's dynamics on the tracks, but since then has evolved. Lastly the PreScan from Siemens is for commercial use, where they offer services of hands-on training and support packages to help the client and also the possibility to use the cloud service provided by Siemens itself.

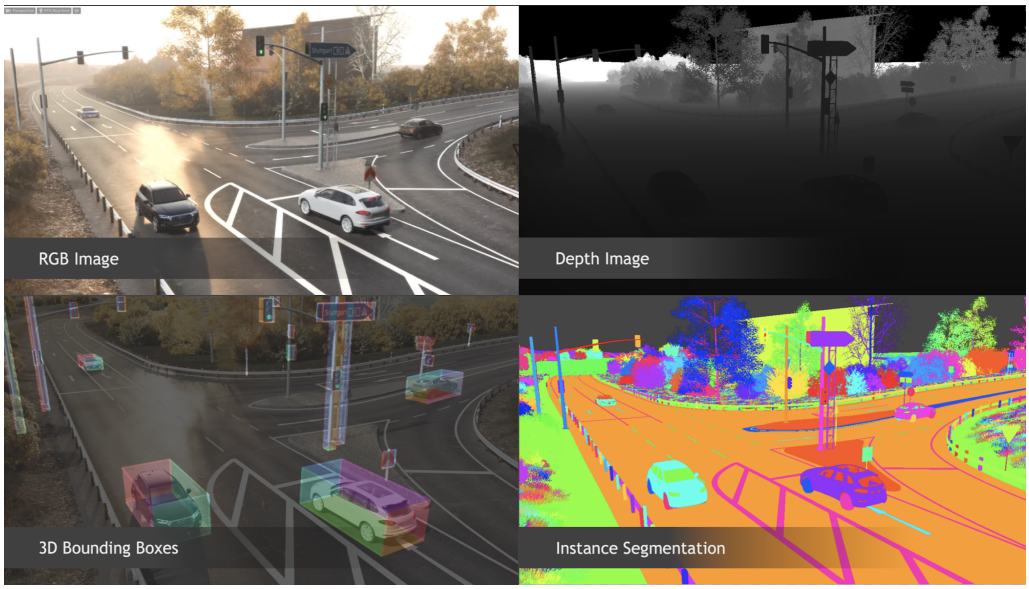


Figure 3.17: Sample of NVIDIA DRIVE Sim sensors

3.3.1 Relevant works done using driving simulators

The autonomous driving simulators are multi-purpose tools that can be used for research purposes. Hu et al. [21], proposed an Unsupervised Domain Adaptation (UDA) approach using adversarial discriminative and generative methods in synthetic generated data, for lane detection and classification in autonomous driving applications. The overall UDA action is to take a synthetic dataset with labels of the source domain and leverage the power of adversarial generative and feature discriminators to predict the lane location and class in the target domain, which in this case is the real world. Furthermore, the authors present a dataset generator called Simulanes, capable of creating synthetic datasets with photo-realistic traffic scenarios and a variety of weather conditions using the CARLA simulator.

Additionally, Pahnl et al. [32] conduct research to showcase the potential of using Generative Adversarial Networks [17] (GANs) in autonomous drive simulation, not just transforming images between domains, but also for conducting driving tests. The authors used a Dual Contrastive Learning Generative Adversarial Networks [19] (DCLGAN) to convert images from the CARLA simulator into realistic images, that are then used to evaluate the effect of domain adaptation, from synthetic to real, focusing on a Lane Keeping Assist System (LKAS).

In addition to the previous papers, research made by Malayjerdi et al. [30] in the area of safety evaluation of autonomous vehicles, to develop advanced techniques using simulators for their realistic and immersiveness characteristics. For this, the integration of simulators is crucial, as it allows to evaluate autonomous vehicles decisions and safety in different situations. The authors, gathered geospatial data collected from drones to then be converted into a 3D map applicable to the LGSVL simulator, which was later tested by autonomous vehicles.

Nowadays, the high-fidelity and photo-realistic graphics that video games have, can serve as a

valuable research tool, even though they lack built-in annotations. In this topic, Richter et al. [38] propose an approach of creating pixel-accurate semantic label maps for images extracted from these video games. Typically the source code of these games or game engines is not accessible to the public, so they show an alternative method by analyzing and extracting information between the game objects renders and the hardware, using tools that act as a graphics debugger like RenderDoc [24]. Using this information, it is possible to identify and isolate the functions calls that are responsible for rendering the objects, enabling the identification of each object individually and create a label map.

Johnson-Roberson et al. [23] also propose a method of generating a synthetic dataset using video games, by capturing screenshots of the game, depth image and auxiliary information that was stored in the graphics processing unit (GPU). The depth and auxiliary data are retrieved directly from Direct3D, which is an API made to render three-dimensional graphics usually in games. After completing the dataset, it is used train neural networks for vehicle detection and compared to other datasets, such as Cityscapes [11] and KITTI [15] using similar neural network architectures. This method led to an improvement the performance.

Talwar et al. [45] also conducted a study to determine if models trained with synthetic data can perform well in real-world scenarios. For this study, datasets were collected from multiple simulators, two different versions from LGSVL simulator, as well as KITTI and Waymo [2] to serve as a real-world dataset. Object detection models were trained with the two synthetic datasets and the KITTI dataset, and then tested with all four datasets, as illustrated in the Fig. 3.18.

Mean Average Precision (mAP) for training set/test set pairs				
	Test Set			
Training Set	LGSVL Sim1	LGSVL Sim2	KITTI	Waymo
LGSVL Sim1	87.75%	56.40%	34.93%	17.11%
LGSVL Sim2	53.61%	92.92%	15.73%	19.2%
KITTI	25.64%	2.42%	68.67%	13.27%

Figure 3.18: Results of the study. [45]

In terms of results, the models trained on synthetic datasets had a better average performance when compared to the model trained on KITTI, and both of them also performed better on the Waymo real-world dataset.

Chapter 4

Synthetic Dataset

In general, a dataset is a collection of data that is organized and structured for a specific purpose, and is an essential part of any machine learning, research or data analysis project, as it contains raw information that will be used to collect insights, draw conclusions and make decisions.
In this Chapter, it will be explained in detail how the synthetic dataset was constructed and its characteristics. A thorough discussion of the features, methodologies and any relevant background information, which could be of paramount importance to understand the context and importance of the data.

4.1 CARLA Simulator

As mentioned in the subsection 3.3, CARLA [13] is an open-source autonomous vehicle simulator developed by the Computer Vision Center in Barcelona in Spain.
It provides a realistic simulation of urban environments, including detailed models of vehicles, pedestrians, buildings, and other objects. It also includes features such as dynamic weather conditions, traffic lights, and pedestrian crossings.
One of the main reasons to use CARLA is the possibility of testing autonomous driving systems in a controlled environment without any risk of damaging or injuring people or property, therefore making it an ideal tool for evaluating and improving the performance of autonomous vehicle systems.

4.1.1 Towns

CARLA is composed of multiple maps, called towns, examples in Fig. 4.1, that are the 3D representations of the simulated world scenery. Each town follows its own landscape ideology using the environment objects, lanes composition and textures, the available towns are the following:

- **Town01**: A basic town layout consisting of "T junctions". [13]
- **Town02**: Similar to Town01, but smaller. [13]

- **Town03**: The most complex town, with a 5-lane junction, a roundabout, unevenness, a tunnel, and more. [13]
- **Town04**: An infinite loop with a highway and a small town. [13]
- **Town05**: Squared-grid town with cross junctions and a bridge. It has multiple lanes per direction. Useful to perform lane changes. [13]
- **Town06**: Long highways with many highway entrances and exits. It also has a Michigan left. (A U-turn that allows a vehicle to join the other side of the road in the opposite direction) [13]
- **Town07**: A rural environment with narrow roads, barns and hardly any traffic lights. [13]
- **Town10**: A city environment with different environments such as an avenue or promenade, and more realistic textures. [13]

4.1.2 Weathers

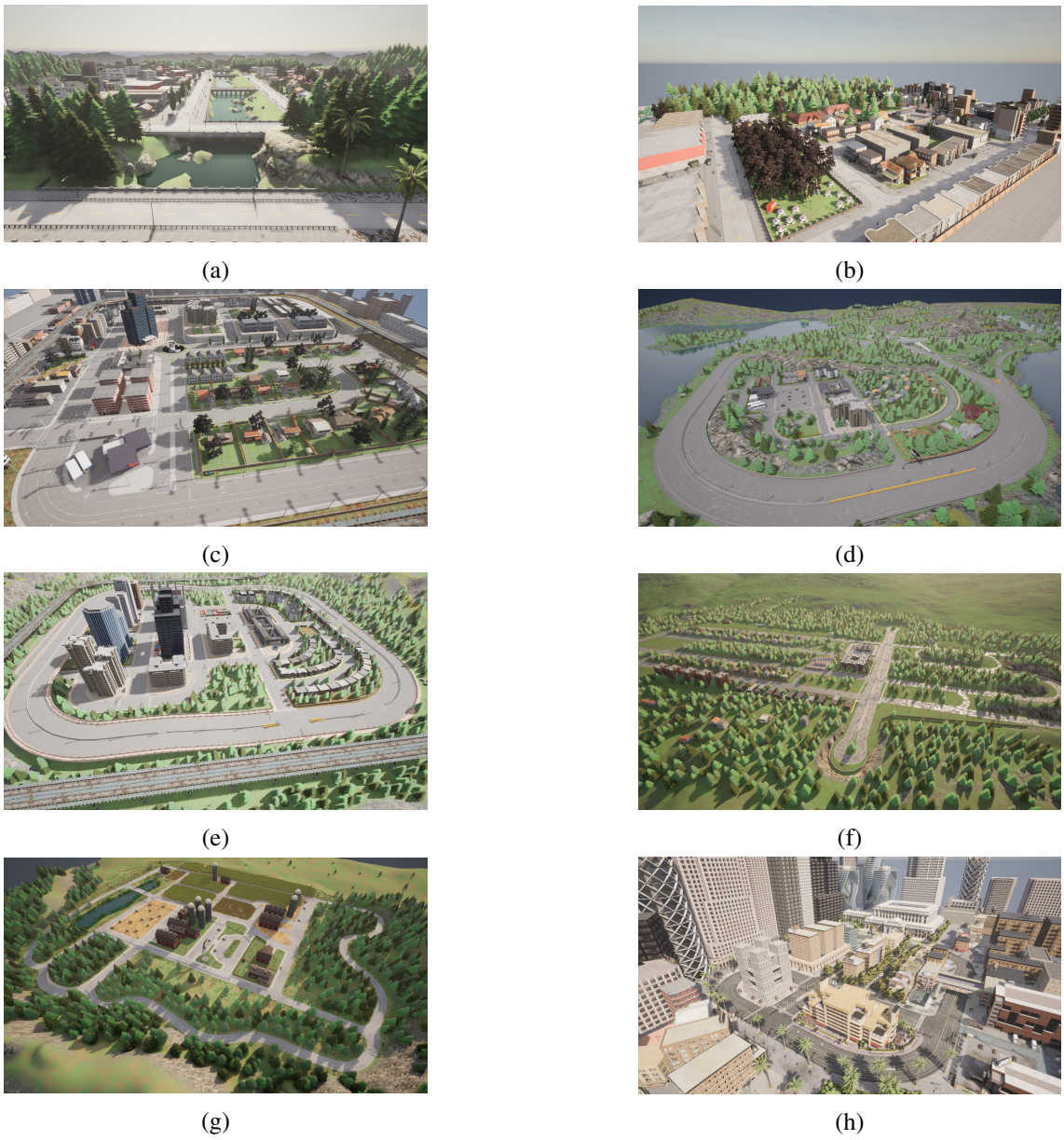
CARLA has dynamic weather customization with multiple parameters available, for example, cloudiness, precipitation, wind intensity and many more. CARLA also provides predefined weather conditions that vary from two different variables, three different times of day and seven types of weather, forming 21 different combinations, see Table 4.1.

Table 4.1: All predefined weather combinations from CARLA

	Noon	**Sunset**	**Night**
Clear	Clear Noon	Clear Sunset	Clear Night
Cloudy	Cloudy Noon	Cloudy Sunset	Cloudy Night
Hard Rain	Hard Rain Noon	Hard Rain Sunset	Hard Rain Night
Mid Rain	Mid Rain Noon	Mid Rain Sunset	Mid Rain Night
Soft Rain	Soft Rain Noon	Soft Rain Sunset	Soft Rain Night
Wet Cloudy	Wet Cloudy Noon	Wet Cloudy Sunset	Wet Cloudy Night
Wet	Wet Noon	Wet Sunset	Wet Night

4.1.3 Data Acquisition

CARLA follows the client-server architecture, where the world itself and its configurations are the server, and the client is able to connect to this world, modify it and participate in it with its autonomous or manual driving model.
The developers of the simulator provided through GitHub [10], the open-source code for the entire application and some code examples. In these examples were present some files that are able to



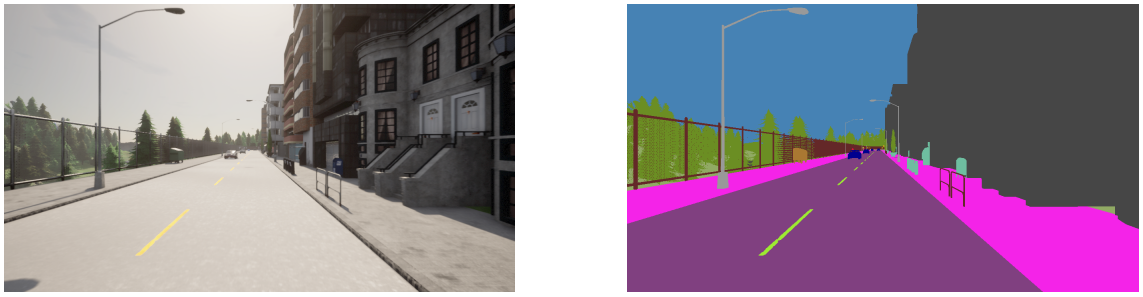
(a) (b) (c) (d) (e) (f) (g) (h)

Figure 4.1: All the towns available in CARLA. (a) Town01. (b) Town02. (c) Town03. (d) Town04. (e) Town05. (f) Town06. (g) Town07. (h) Town10.

connect to a CARLA server and instantiate a car, modify the car model, alter to whichever weather mentioned in 4.1.2, choose what kind of sensors to use, record images directly and many other options. The one thing needed that did not come with the source code was the ability to record two different type of data at the same time, so we had the develop that feature in order to be able to record the RGB image and the semantic segmentation image of the same frame.

The CARLA simulator provides multiple usable sensors to extract the kind of sensory data the users want, in the scope of this thesis CARLA does not provide separately the image labels for the road and road line segmentation like the BDD100K provides. So to fix this issue it was chosen the semantic segmentation images, in Cityscapes style, because it already contains classes for the

road and the road line that can be extracted later, for example seen in Fig. 4.2.



(a) (b)

Figure 4.2: Example of two different type of data recording the same frame. (a) RGB sensor image. (b) Semantic segmentation sensor image in cityscapes style.

To build a world that reassembles reality, there is a need to add other actors so the vehicle can interact with it, because the car in a real-world scenario being driven by the autonomous driving model is not by itself in the roads, coexists with other vehicles and pedestrians. Thus CARLA provides the possibility to create these agents to interact with the world itself, see Fig. 4.3. Increasing the number of agents leads to more computational power necessary to maintain a good fps rate, so due to the computer limitations, the maximum of vehicles and pedestrians used were 80 and 40 respectively.

Figure 4.3: World filled with autonomous pedestrians and vehicles

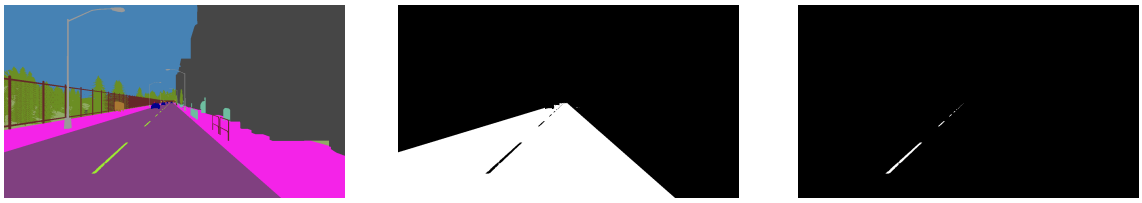
Another important note is to be careful with the impact of the climate/weather changes in the synthetic world, since it can affect the visibility of the car. For example, during the night, the vehicles must have the lights turned on, for the fact that if they don't the visibility decreases drastically as can be seen in the Figs. 4.4 and 4.5.

(a) (b)

Figure 4.4: (a) POV of the car RGB sensor with no lights. (b) POV of the car RGB sensor with lights.

4.1.4 Data Labeling

Initially, the idea was to search for simulators or synthetic datasets publicly available, that have the same kind of annotations for drivable area and lane line tasks as the BDD100k dataset.
Given these challenges, it was decided to take advantage of the semantic segmentation sensory data from CARLA, as mentioned in 4.1.3. This involved extracting every pixel that belongs to the 'Road' and 'Road Line' classes from each image, and generate two separate label images, one for each class respectively shown in Fig. 4.6.



(a) (b) (c)

Figure 4.6: (a) Semantic segmentation image. (b) Road image mask. (c) Road line image mask.

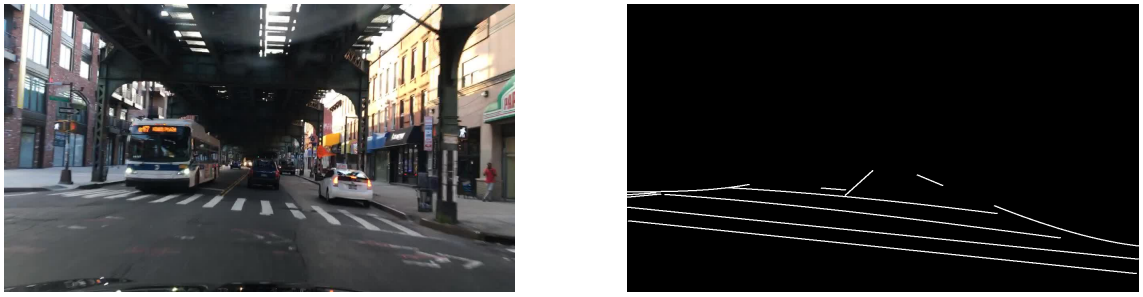
By generating the image labels automatically, it was able to save a significant amount of time and also view this challenge from a different perspective, as a transfer learning and domain adaptation problem. In this way, the models will be tested on their ability to transfer the training they received on a synthetic dataset to a real dataset with diverse tasks. While the drivable area may not experience significant changes, the road lines will differ greatly due to the fact that each dataset labels road lines differently. In CARLA, every white road line is marked, as shown in Fig. 4.7. In contrast, the BDD100k dataset uses continuous marks to divide lanes and crosswalk boundaries, as shown in Fig. 4.8."

(a) (b)

Figure 4.5: (a) POV of the car RGB sensor and other vehicles with no lights. (b) POV of the car RGB sensor and other vehicles with lights.

(a) (b)

Figure 4.7: Demonstration of the CARLA road line masks characteristics



(a) (b)

Figure 4.8: Demonstration of the BDD100K road line masks characteristics

4.2 Dataset Composition

To ensure an even distribution of images across all weather conditions and towns and to match the size of the BDD100k dataset, tried to divide the dataset as evenly as possible. This resulted in 476 images for each combination of weather and town, as shown in Table 4.2.

The goal of evenly distributing the dataset was to avoid the risk of the models being affected by bias that might be caused by certain towns or weather conditions being overrepresented.
To split the dataset the same way as the BDD100k dataset, the test and validation sets need to

Table 4.2: Synthetic CARLA dataset composition. Each numerical value represents the number of images.

Weathers/Maps	Town01	Town02	Town03	Town04	Town05	Town06	Town07	Town10	Total
Clear Night	476	476	476	476	476	476	476	476	3808
Clear Noon	476	476	476	476	476	476	476	476	3808
Clear Sunset	476	476	476	476	476	476	476	476	3808
Cloudy Night	476	476	476	476	476	476	476	476	3808
Cloudy Noon	476	476	476	476	476	476	476	476	3808
Cloudy Sunset	476	476	476	476	476	476	476	476	3808
Hard Rain Night	476	476	476	476	476	476	476	476	3808
Hard Rain Noon	476	476	476	476	476	476	476	476	3808
Hard Rain Sunset	476	476	476	476	476	476	476	476	3808
Mid Rain Night	476	476	476	476	476	476	476	476	3808
Mid Rain Noon	476	476	476	476	476	476	476	476	3808
Mid Rain Sunset	476	476	476	476	476	476	476	476	3808
Soft Rain Night	476	476	476	476	476	476	476	476	3808
Soft Rain Noon	476	476	476	476	476	476	476	476	3808
Soft Rain Sunset	476	476	476	476	476	476	476	476	3808
Wet Cloudy Night	476	476	476	476	476	476	476	476	3808
Wet Cloudy Noon	476	476	476	476	476	476	476	476	3808
Wet Cloudy Sunset	476	476	476	476	476	476	476	476	3808
Wet Night	476	476	476	476	476	476	476	476	3808
Wet Noon	476	476	476	476	476	476	476	476	3808
Wet Sunset	476	476	476	476	476	476	476	476	3808
Total	9996	9996	9996	9996	9996	9996	9996	9996	79968

be around 10000 images each and the train set the remaining 60000 images. Instead of taking a portion from each town and weather condition for each set, it was decided to reserve certain towns for specific purposes. For example, Town03 and Town05 were reserved for the validation and test sets, which were each divided in half and distributed to each set, resulting in approximately 9996 images each. The remaining towns were assigned to the training set, totaling 59976 images. This approach was taken in order to expose the models to completely new scenarios during validation and testing that they had not encountered during training.

- **Training set:** Town01,02,04,06,07,10 = 59976 images
- **Validation set:** Half Town03,05 = 9996 images
- **Testing set:** Half Town03,05 = 9996 images

Chapter 5

Methodology

5.1 Training

The training phase consists of two parts, first training the models using the synthetic dataset made from CARLA, mentioned in Chapter 4, and then train the exact same models with the BDD100K dataset.

5.1.1 Models

In this research, the focus is on the recent multi-task models YOLOP and HybridNets, which have been thoroughly discussed in Section 3.2. These models are specifically designed and structured to perform drivable area segmentation and lane segmentation with state-of-the-art results. The main difference of the two networks, besides the different encoder models, is that YOLOP uses a two-head approach, one for each task, while HybridNets uses a single-head approach. So, to make the results more comparable between the models, the decision was made to modify the HybridNets to use the same two-head approach of YOLOP.

5.1.2 Synthetic Training

The synthetic dataset generated using CARLA, as detailed in Chapter 4, will be used to train the previously mentioned models. The training settings for these models include a duration of 40 epochs, as it will be later demonstrated that is more than sufficient for the models to stabilize. The Adam [25] optimizer is employed with the β_1 and β_2 set to their standard values of 0.9 and 0.999, and a learning rate of 0.0001.

5.1.3 Transfer Learning and Domain Adaptation

The transfer learning and domain adaptation will be performed on the models, that already have been trained with the synthetic dataset. This is because, during the synthetic training, the models learned how to detect the entire of road and its markings, and in the BDD100K dataset the models

will learn how to detect only their side of the road, with or without the ego-lane distinction, and detect the lane lines.

5.1.4 BDD100K Dataset

The BDD100K is an open dataset designed to support multi-task autonomous driving research. The dataset originally was compiled by collecting over 100,000 videos of real-world driving, providing this way realistic driving scenarios. BDD100K focuses on ten distinct tasks:

- Drivable area segmentation
- Lane detection
- Semantic segmentation
- Instance segmentation
- Multi-object segmentation tracking
- Multi-object detection tracking
- Image tagging
- Road object detection
- Imitation learning
- Domain adaptation

The dataset videos are divided into 70K for training, 10K for validation and 20K for testing. The 10^{th} frame of each video is extracted and annotated for all the tasks to compose the image dataset. Additionally, the test split labels are not public, so the dataset splits had to be rearranged. The original validation split of 10K images became the test set, then 10K images were randomly selected from the train split to become the validation set, and the remaining 60K images stayed as the train split.
Furthermore, the lane annotations from the dataset were divided into three sub-classes: lane categories (9 classes), lane directions (3 classes) and lane styles (3 classes). So, instead of transforming these annotations into typical labels of lane dividers, decided to take advantage and use the labels provided by the authors of YOLOP.
We opted to use the BDD100K dataset because it is one of the few multipurpose datasets that includes the drivable area and lane detection tasks. Furthermore, we chose to use a single multipurpose dataset instead of two datasets, each one being specific for one task, because the synthetic dataset used for the first phase of the training is also multipurpose, and we wanted to ensure this consistency.

5.1.5 Loss Functions

For a two-head architecture, where each head is responsible for one task, a Binary Cross Entropy (BCE) loss is used for each one and then sum both of them with equal weight to form the total loss, demonstrated in the Equation 5.1.

$$TotalLoss = BCE\_loss\_drivable\_area\_head + BCE\_loss\_lane\_head \tag{5.1}$$

For a one-head architecture, responsible for segmenting the drivable area and the lanes, the Cross Entropy (CE) Loss is used.
Whereas in the other configuration of a two-head architecture, where the task of the drivable area is done by considering the ego-lane, the Cross Entropy Loss is combined with the Binary Cross Entropy Loss for the lane segmentation, as seen in the Equation 5.2.

$$TotalLoss = CE\_loss\_drivable\_area\_head + BCE\_loss\_lane\_head \tag{5.2}$$

Later on, the loss correspondent to the lane lines was replaced with the Iou loss, represented in the Equation 2.6. So this led to updates in the total losses for both non ego-lane and ego-lane, demonstrated in the Equations. 5.3 and 5.4 respectively.

$$TotalLoss = BCE\_loss\_drivable\_area\_head + IoU\_loss\_lane\_head \tag{5.3}$$

$$TotalLoss = CE\_loss\_drivable\_area\_head + IoU\_loss\_lane\_head \tag{5.4}$$

Chapter 6

Results and Discussion

6.1 Synthetic Results

The results of the training of both models on the synthetic dataset went smoothly, as seen by the early convergence of their loss functions at 10 epochs, as illustrated in Figs. 6.1 and 6.2. The validation and test results, in the Tables 6.1 and 6.2, prove the training of the models was successful. It is also worth noting that both results are relatively similar, as expected taking into account what was mentioned in the Chapter 4.2, that the validation and test subsets are composed of the same towns. The results also highlight an issue with the pixel accuracy metric. Even though it shows 100% accuracy in the test subset for the HybridNets model in the task road line segmentation, it also shows 69% of mean IoU between all classes. This discrepancy happens due to the fact that the model predicts most of the pixels as the background class rather than the lane line. Additionally, most of the lane line images contain only a small number of pixels that belong to the lane line class, often less than 1%. Therefore, the pixel accuracy metric is not the most fitting metric for evaluating lane segmentation and image segmentation in general.
The performance between both models in the test set is quite similar, especially in the road segmentation task. However, in the lane line segmentation task, the YOLOP obtained an 83% in mIoU against the 69% from HybridNets, giving the assurance that the YOLOP model is more capable that the HybridNets model on this two-head approach. As an illustration the Figs. 6.3 and 6.4, show an example of the synthetic dataset and its correspondence output from each model.

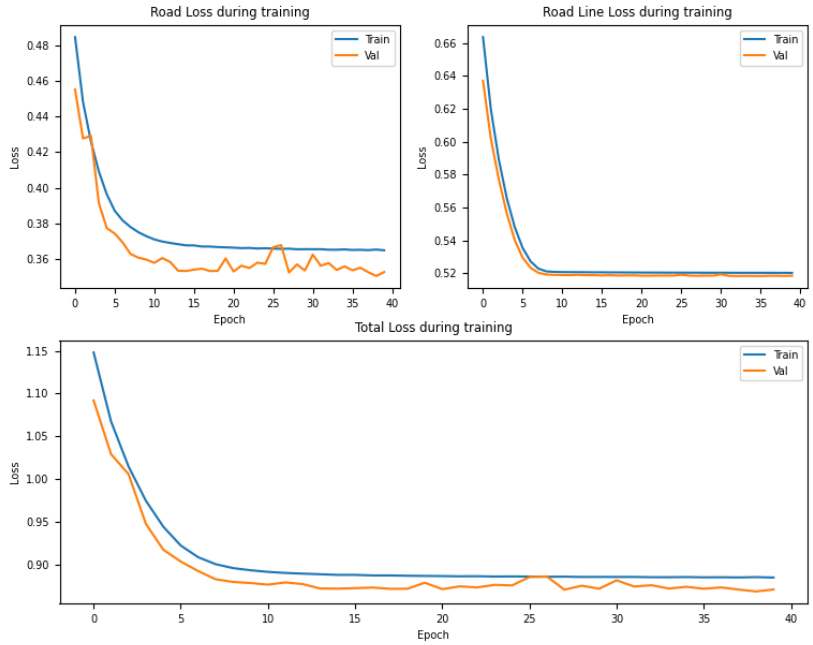


Figure 6.1: YOLOP losses during synthetic training.

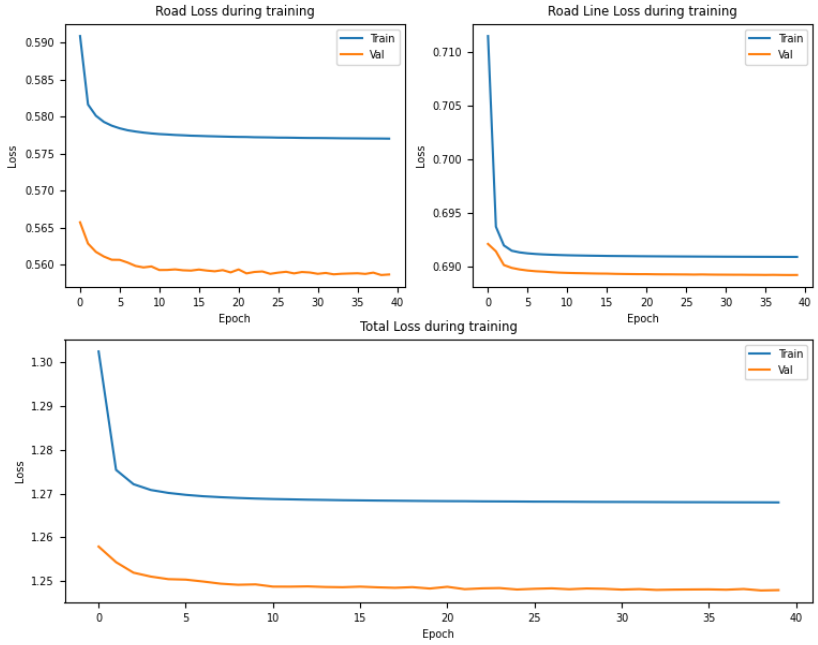


Figure 6.2: HybridNets losses during synthetic training.

Table 6.1: Best validation results during synthetic training.

(a) Road.

Models\Metrics	Pixel Accuracy (%)	mIoU (%)	F1-Score (%)
YOLOP	99	97	99
HybridNets	99	97	98

(b) Road line.

Models\Metrics	Pixel Accuracy (%)	mIoU (%)	F1-Score (%)
YOLOP	100	85	92
HybridNets	100	72	84

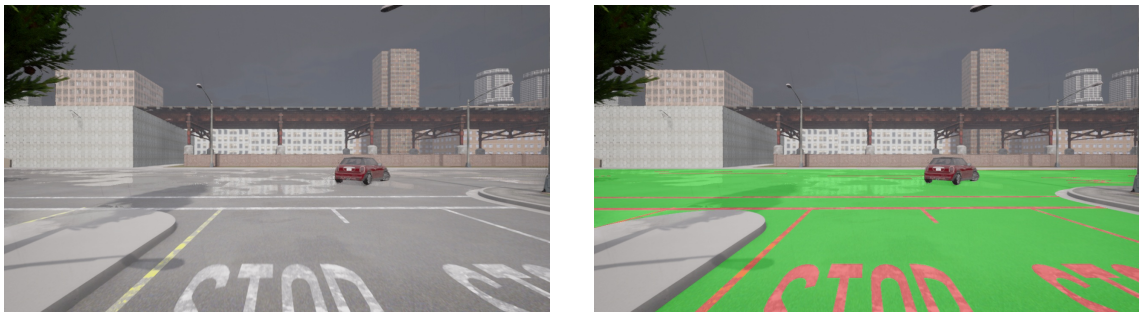
Table 6.2: Test results after synthetic training.

(a) Road.

Models\Metrics	Pixel Accuracy (%)	mIoU (%)	F1-Score (%)
YOLOP	99	97	98
HybridNets	99	97	98

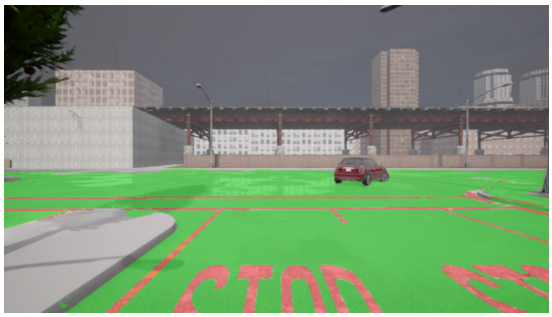
(b) Road line.

Models\Metrics	Pixel Accuracy (%)	mIoU (%)	F1-Score (%)
YOLOP	100	83	90
HybridNets	100	69	81

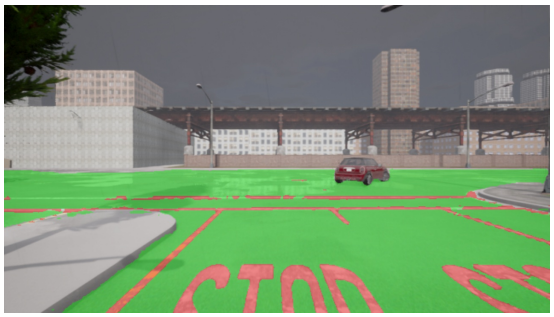


(a) Real image. (b) Segmented image.

Figure 6.3: Demonstration of an image from the synthetic dataset and its correspondence label image.



(a) YOLOP output image.



(b) HybridNets output image.

Figure 6.4: Demonstration of the YOLOP and HybridNets segmentations on the synthetic dataset.

6.1.1 Weather Influence

The synthetic dataset was generated with multiple weather conditions and at different times of the day. This provides the opportunity to study the performance of the models under different conditions and understand how they are affected.
The results in the Tables 6.3 and 6.4, indicate that both models are able to maintain the performance through the different times of day. The Tables 6.5 and 6.6, show that the models have the same performance during different weathers conditions, demonstrating their ability to generalize at different weather conditions and times of day.

Table 6.3: Test results of YOLOP on different times of day.

(a) Road.

Time of Day\Metrics	mIoU (%)	F1-Score (%)
Noon	97	99
Sunset	97	99
Night	97	98

(b) Road line.

Time of Day\Metrics	mIoU (%)	F1-Score (%)
Noon	83	90
Sunset	83	90
Night	84	91

Table 6.4: Test results of HybridNets on different times of day.

(a) Road.

Time of Day\Metrics	**mIoU (%)**	**F1-Score (%)**
Noon	97	98
Sunset	97	98
Night	97	98

(b) Road line.

Time of Day\Metrics	**mIoU (%)**	**F1-Score (%)**
Noon	68	81
Sunset	68	81
Night	69	81

Table 6.5: Test results of YOLOP on different weathers.

(a) Road.

Type of Weather\Metrics	**mIoU (%)**	**F1-Score (%)**
Clear	97	99
Cloudy	98	99
Hard Rain	97	98
Mid Rain	96	98
Soft Rain	97	99
Wet Cloudy	97	98
Wet	97	98

(b) Road line.

Type of Weather\Metrics	**mIoU (%)**	**F1-Score (%)**
Clear	83	90
Cloudy	84	91
Hard Rain	85	91
Mid Rain	84	91
Soft Rain	82	89
Wet Cloudy	85	91
Wet	82	89

Table 6.6: Test results of HybridNets on different weathers.

(a) Road.

Type of Weather\Metrics	mIoU (%)	F1-Score (%)
Clear	97	98
Cloudy	97	98
Hard Rain	97	98
Mid Rain	97	98
Soft Rain	97	98
Wet Cloudy	97	98
Wet	97	98

(b) Road line.

Type of Weather\Metrics	mIoU (%)	F1-Score (%)
Clear	69	81
Cloudy	68	80
Hard Rain	70	80
Mid Rain	69	81
Soft Rain	67	79
Wet Cloudy	70	82
Wet	67	79

6.2 Transfer Learning and Domain Adaptation Results

Now, with the models already trained and tested with the synthetic dataset, it was proceeded with the training with the real dataset. By observing the losses of both training sessions, represented in the Figs. 6.5 and 6.6, it shows that the models are already stabilizing during the training.

Table 6.7: State-of-the-art test results on the real dataset.

(a) Drivable area.

Models\Metrics	mIoU (%)
YOLOP	91.5
HybridNets	90.5

(b) Lane line.

Models\Metrics	mIoU (%)
YOLOP	26.2
HybridNets	31.6

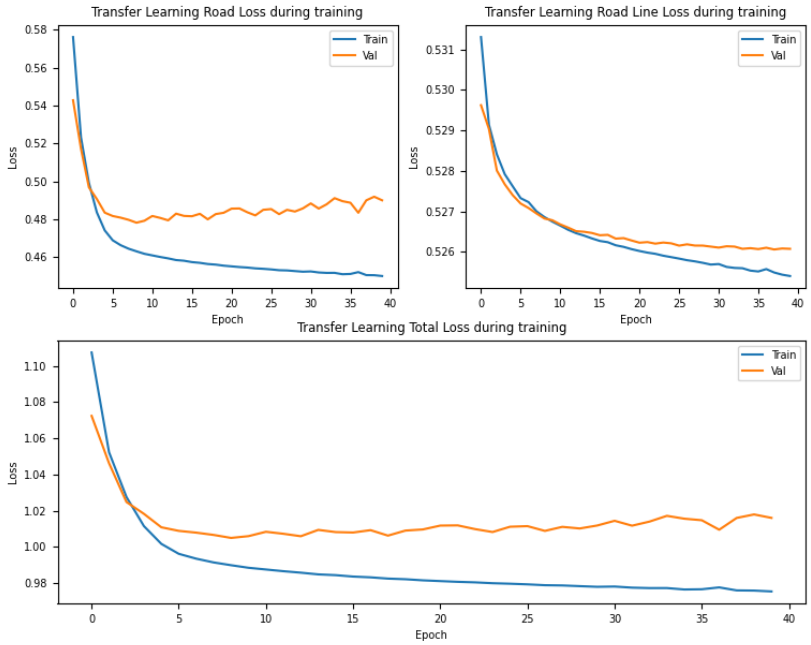


Figure 6.5: YOLOP losses during real training.

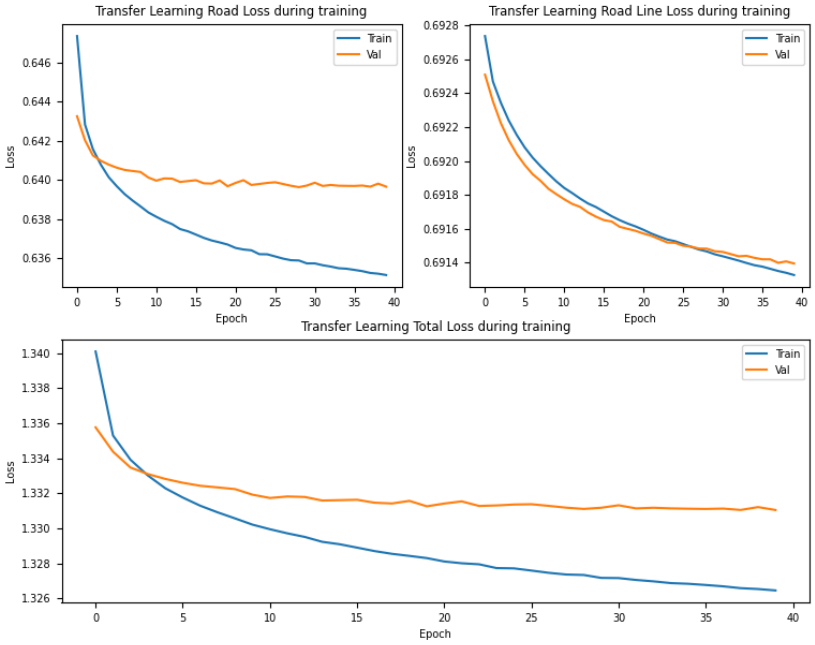


Figure 6.6: HybridNets losses during real training.

The results of the models in the drivable area segmentation were promising, as seen in the Tables 6.8 and 6.9. However, in case of the lane lines task, there was a bit of a discrepancy between the validation and test results. Additionally, when comparing to the state-of-the-art results, in the Table 6.7, the models fell a bit short, especially the HybridNets model.

Table 6.8: Best validation results during real training.

(a) Drivable area.

Models\Metrics	mIoU (%)	F1-Score (%)
YOLOP	85	92
HybridNets	87	93

(b) Lane line.

Models\Metrics	mIoU (%)	F1-Score (%)
YOLOP	45	62
HybridNets	43	60

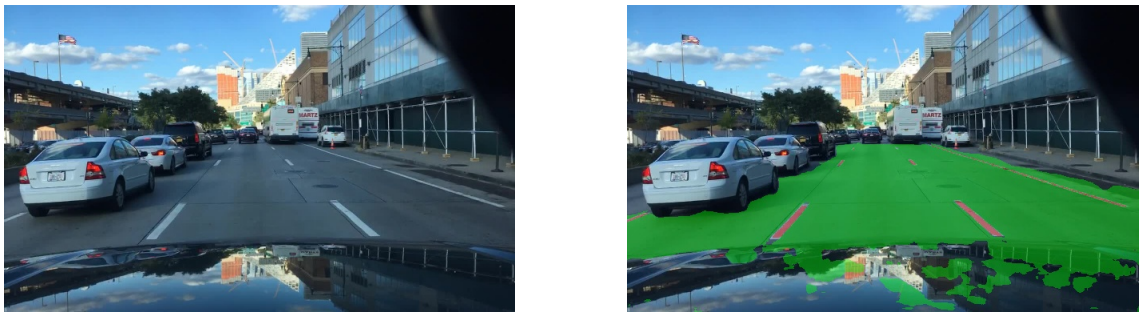
Table 6.9: Test results after real training.

(a) Drivable area.

Models\Metrics	mIoU (%)	F1-Score (%)
YOLOP	85	92
HybridNets	86	93

(b) Lane line.

Models\Metrics	mIoU (%)	F1-Score (%)
YOLOP	24	38
HybridNets	17	28



(a) Real image. (b) Segmented image.

Figure 6.7: Demonstration of the YOLOP segmentation on real dataset.

(a) Real image. (b) Segmented image.

Figure 6.8: Demonstration of the HybridNets segmentation on real dataset.

Looking at the output examples, illustrated in the Figs. 6.7 and 6.8, the models seem to have a problem transitioning from the lane lines of the synthetic dataset to the real dataset. This is because the models still have the tendency to detect all the marks on the road instead of detecting the lines of each lane present.
To possibly solve this issue, it was decided to adapt a loss function in a way to penalize the false positives and false negatives cases into the lane line segmentation task, and so ended up incorporating the IoU loss, see Equations 5.3 and 5.4, proposed by the YOLOP [52] authors.

Table 6.10: Test results after real training with IoU loss.

(a) Drivable area.

Models\Metrics	**mIoU (%)**	**F1-Score (%)**
YOLOP	85	92
HybridNets	86	93

(b) Lane line.

Models\Metrics	**mIoU (%)**	**F1-Score (%)**
YOLOP	24	38
HybridNets	17	28

The results using the IoU loss, presented in the Table 6.10, show no improvement at all in the performance of the models. Thus, after multiple tentatives of adjusting losses, learning rates, optimizers, and other parameters, the best result was obtained, shown in the Table 6.11. These results were obtained using the same BCE loss for the drivable area task, and the IoU loss for the lane lines task and using the Adam optimizer with a learning rate of 0.001, but still not enough improvement to surpass the state-of-the-art results.
Although, the lane lines segmentation task struggles to properly adapt to the new task, this issue is more likely to be a problem from the limitation of the transfer learning, rather than the domain adaptation. Due to the fact that, the models are able to perform well the drivable area task in real-world scenarios.

Table 6.11: Best test results on the real dataset.

(a) Drivable area.

Models\Metrics	mIoU (%)	F1-Score (%)
YOLOP	86	92
HybridNets	86	92

(b) Lane line.

Models\Metrics	mIoU (%)	F1-Score (%)
YOLOP	24	39
HybridNets	18	30

6.3 Non Ego-Lane vs Ego-Lane

To further study the behavior of the models, it was decided to perform the transfer learning and domain adaptation with ego-lane distinction on the drivable area task, results are shown in the Table 6.12.

Even though the performance of the models in the drivable area decreased significantly, the results illustrated in the Figs. 6.9 and 6.10 show promising results, since the models can accurately detect and distinguish the cars ego-lane between the others available lanes. This drop in performance of the models, especially in the drivable area task, is expected due to the increased number of classes that are segmented.

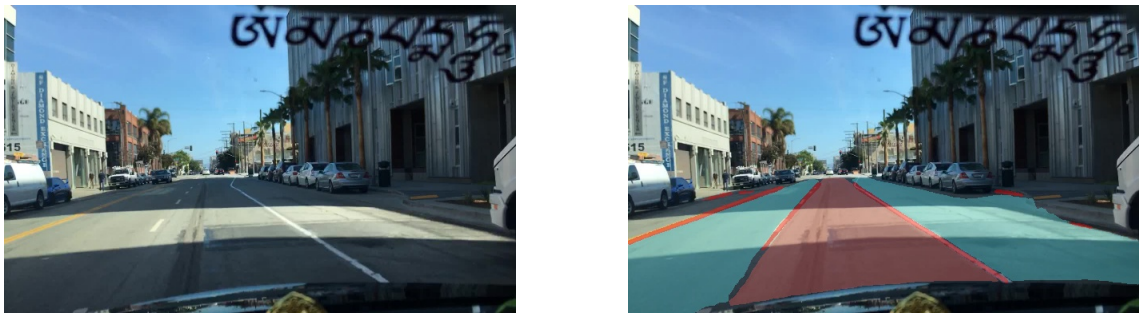
Table 6.12: Test results on the real dataset with ego-lane.

(a) Drivable area with Ego Lane.

Models\Metrics	mIoU (%)	F1-Score (%)
YOLOP	54	93
HybridNets	54	93

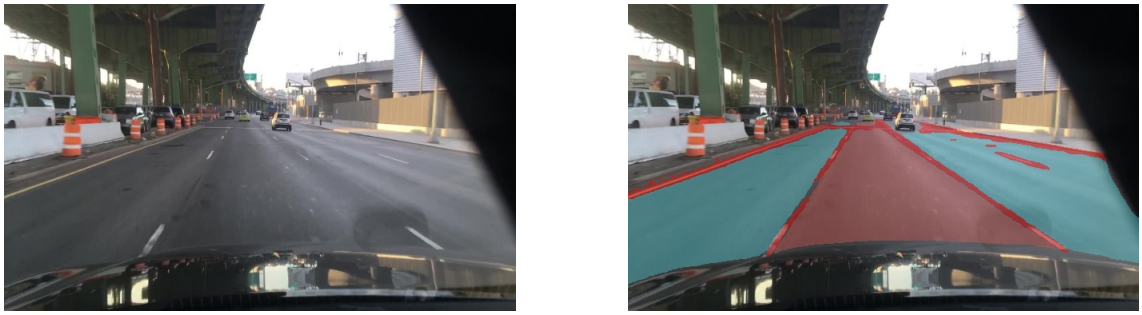
(b) Lane line.

Models\Metrics	mIoU (%)	F1-Score (%)
YOLOP	23	38
HybridNets	20	33



(a) Real image. (b) Segmented image.

Figure 6.9: Demonstration of the YOLOP segmentation on real dataset with ego-lane.



(a) Real image. (b) Segmented image.

Figure 6.10: Demonstration of the HybridNets segmentation on real dataset with ego-lane.

6.4 Multi Head vs Single Head

Finally, this last experiment decided to study the performance of each model, using a single-head approach. The Tables 6.13 and 6.14 present the test results of the trained models on the synthetic and real datasets, respectively. Comparing the results with the multi-head configuration the HybridNets demonstrates an advantage in using the single-head approach over the YOLOP model. This aligns with the original HybridNets architecture, which was built to perform all segmentation tasks with a single head. Even though, it is not with ego-lane detection, the same decrease in performance happens for the same reason. Using a single-head configuration means that the network will have to determine which class each pixel belongs to, between more classes compared to each one of the heads from the multi-head approach.

Table 6.13: Test results on the synthetic dataset with one head models.

(a) Drivable area.

Models\Metrics	**mIoU (%)**	**F1-Score (%)**
YOLOP	98	99
HybridNets	97	99

(b) Lane line.

Models\Metrics	**mIoU (%)**	**F1-Score (%)**
YOLOP	85	92
HybridNets	75	85

Table 6.14: Test results on the real dataset with one head models.

(a) Drivable area.

Models\Metrics	**mIoU (%)**	**F1-Score (%)**
YOLOP	60	97
HybridNets	61	97

(b) Lane line.

Models\Metrics	**mIoU (%)**	**F1-Score (%)**
YOLOP	23	38
HybridNets	23	37

Chapter 7

Conclusions

The autonomous driving industry goal is to create vehicles capable of driving safely and independently without any human control. During the development of these driving systems, one of the biggest challenges they face is how to properly test the vehicle's performance and behavior, in order to know that are ready to be deployed into the real world. The simulators allow to train and test the autonomous driving systems in a controlled risk-free environment. The simulators also enable testing these systems in multiple scenarios, from the most dangerous, to the most typical ones without any safety being compromised. Another substantial benefit of the simulators is their ability to allow fast prototype and iteration through autonomous driving systems, and consequently being more faster and more efficient to deploy them into the real world. Furthermore, to train autonomous driving models require a lot of data, and to capture this kind of data can be very costly and logistically difficult, and so the simulators can generate this kind of data with labels ready instantly, speeding up the process even more.

This dissertation carried out an research about the usage of simulators in the autonomous driving scope, to be more specific, how the synthetic data generated from the CARLA [13] simulator with state-of-the-art models, YOLOP [52] and HybridNets [50], on the drivable area and lane line segmentation tasks in real-world scenarios. Other additional comparison studies were conducted on how the models would perform with ego-lane distinction in the drivable area segmentation task, and using different network head approaches, single and multi-head.

In terms of results, both models could not surpass the state-of-the-art results, YOLOP achieving 86% and 24% mIoU in the drivable area and lane line segmentation respectively, HybridNets achieving 86% and 18% in mIoU, both with multi-head architecture. In the ego-lane distinction, both models dropped the drivable area performance to 54% and in the lane line achieving 56% and 20%, YOLOP and HybridNets respectively. For the different head architectures, the two models obtained better performance in most of the tasks using the multi-head approach, which is understandable for the fact that it is much easier for a network to learn how to segment between two classes in each head, than compared to three or more classes in a single head.

Even though, this research did not get the most ideal results, it was created a synthetic dataset generated from an autonomous driving simulator. Containing information about different combi-

nations of weather and times of day, and also with 22 classes for semantic segmentation available, that could be reused in the future for researches or commercial purposes.

7.1 Future Work

The rise of autonomous driving in conjunction with the rise of deep learning resulted in multiple contributions to the improvement in the area. Only last year, in 2022, it was released both models used in this thesis and in the second half of the year, there was already an improvement for the YOLOP called YOLOPv2 [18].

Although, there was a limited amount of time, there are other approaches that were left out, such as test and train the models with other real or synthetic datasets, try to perform a mix of synthetic-real datasets with different ratios and see how the models would perform, and try other deep learning networks. One advantage of simulated data is the ability to generate it in any quantity that we desire, which raises the question of what could happen if we keep increasing the size of the simulated data compared to a fixed size real dataset. Jaipuria et al. [22] performed a study about the effectiveness of synthetic augmented data with sim2real style transfer in autonomous vehicle vision tasks such as parking slot detection, lane detection and monocular depth. The study's investigation revealed that in case of the parking slot detection, the model trained with 50% of sim2real data in the training data achieved a 30% improvement over the model trained with 100% of real data, and in the lane detection task, the model with 60% of sim2real data performed 40% better compared to the model trained with 100% real data. This shows the different techniques that could be used to improve results and the potential of using synthetic data to fill the gap of real datasets.

References

[1] Tusimple lane detection benchmark. URL: https://github.com/TuSimple/tusimple-benchmark/tree/master/doc/lane_detection,journal={GitHub},author={TuSimple}.

[2] Waymo open dataset: An autonomous driving dataset. URL: https://waymo.com/open/.

[3] Jaccard index, Oct 2022. URL: https://en.wikipedia.org/wiki/Jaccard_index.

[4] R. Adams and L. Bischof. Seeded region growing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(6):641–647, 1994. doi:10.1109/34.295913.

[5] Jose M. Alvarez, Theo Gevers, and Antonio M. Lopez. Road detection by one-class color classification: Dataset and experiments, 2014. URL: https://arxiv.org/abs/1412.3506, doi:10.48550/ARXIV.1412.3506.

[6] Karsten Behrendt and Ryan Soussan. Unsupervised labeled lane markers using maps. In *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, pages 832–839, 2019. doi:10.1109/ICCVW.2019.00111.

[7] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection, 2020. URL: https://arxiv.org/abs/2004.10934, doi:10.48550/ARXIV.2004.10934.

[8] Panpan Cai, Yiyuan Lee, Yuanfu Luo, and David Hsu. Summit: A simulator for urban driving in massive mixed traffic. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4023–4029. IEEE, 2020.

[9] John Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698, 1986. doi:10.1109/TPAMI.1986.4767851.

[10] Carla-Simulator. Carla-simulator/carla: Open-source simulator for autonomous driving research. URL: https://github.com/carla-simulator/carla.

[11] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[12] Nameirakpam Dhanachandra, Khumanthem Manglem, and Yambem Jina Chanu. Image segmentation using k -means clustering algorithm and subtractive clustering algorithm. *Procedia Computer Science*, 54:764–771, 2015. Eleventh International Conference on Communication Networks, ICCN 2015, August 21-23, 2015, Bangalore, India Eleventh International Conference on Data Mining and Warehousing, ICDMW 2015, August 21-23, 2015, Bangalore, India Eleventh International Conference on Image and Signal Processing, ICISP 2015, August 21-23, 2015, Bangalore, India. URL: https://www.sciencedirect.com/science/article/pii/S1877050915014143, doi:https://doi.org/10.1016/j.procs.2015.06.090.

[13] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017.

[14] Epic Games. Unreal engine. URL: https://www.unrealengine.com.

[15] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*, 2013.

[16] R.C. Gonzalez and R.E. Woods. *Digital Image Processing 4th Edition*. Pearson, 2018. URL: https://books.google.pt/books?id=0F05vgAACAAJ.

[17] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. volume 3, page 2672 – 2680, 2014. Cited by: 32638. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84937849144&partnerID=40&md5=6441b2a288c5fdded2adbcc8b21e092c.

[18] Cheng Han, Qichao Zhao, Shuyi Zhang, Yinzi Chen, Zhenlin Zhang, and Jinwei Yuan. Yolopv2: Better, faster, stronger for panoptic driving perception, 2022. URL: https://arxiv.org/abs/2208.11434, doi:10.48550/ARXIV.2208.11434.

[19] Junlin Han, Mehrdad Shoeiby, Lars Petersson, and Mohammad Ali Armin. Dual contrastive learning for unsupervised image-to-image translation, 2021. URL: https://arxiv.org/abs/2104.07689, doi:10.48550/ARXIV.2104.07689.

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. In *Computer Vision – ECCV 2014*, pages 346–361. Springer International Publishing, 2014. URL: https://doi.org/10.1007%2F978-3-319-10578-9_23, doi:10.1007/978-3-319-10578-9_23.

[21] Chuqing Hu, Sinclair Hudson, Martin Ethier, Mohammad Al-Sharman, Derek Rayside, and William Melek. Sim-to-real domain adaptation for lane detection and classification in autonomous driving. In *2022 IEEE Intelligent Vehicles Symposium (IV)*, pages 457–463, 2022. doi:10.1109/IV51971.2022.9827450.

[22] Nikita Jaipuria, Xianling Zhang, Rohan Bhasin, Mayar Arafa, Punarjay Chakravarty, Shubham Shrivastava, Sagar Manglani, and Vidya N. Murali. Deflating dataset bias using synthetic data augmentation. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 3344–3353, 2020. doi:10.1109/CVPRW50498.2020.00394.

[23] Matthew Johnson-Roberson, Charles Barto, Rounak Mehta, Sharath Nittur Sridhar, Karl Rosaen, and Ram Vasudevan. Driving in the matrix: Can virtual worlds replace human-generated annotations for real world tasks?, 2016. URL: https://arxiv.org/abs/1610.01983, doi:10.48550/ARXIV.1610.01983.

[24] Baldur Karlsson. URL: https://renderdoc.org/.

[25] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014. URL: https://arxiv.org/abs/1412.6980, doi:10.48550/ARXIV.1412.6980.

[26] Dong-Gyu Lee. Fast drivable areas estimation with multi-task learning for real-time autonomous driving assistant. *Applied Sciences*, 11(22), 2021. URL: https://www.mdpi.com/2076-3417/11/22/10713, doi:10.3390/app112210713.

[27] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 936–944, 2017. doi:10.1109/CVPR.2017.106.

[28] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(2):318–327, 2020. doi:10.1109/TPAMI.2018.2858826.

[29] Yuanfu Luo, Panpan Cai, Yiyuan Lee, and David Hsu. Gamma: A general agent motion model for autonomous driving. *IEEE Robotics and Automation Letters*, 7(2):3499–3506, 2022. doi:10.1109/LRA.2022.3144501.

[30] Mohsen Malayjerdi, Bariş Cem Baykara, Raivo Sell, and Ehsan Malayjerdi. Safety assessment and simulation of autonomous vehicle in urban environments. *IOP Conference Series: Materials Science and Engineering*, 1140(1):012032, may 2021. URL: https://dx.doi.org/10.1088/1757-899X/1140/1/012032, doi:10.1088/1757-899X/1140/1/012032.

[31] Dong-Hee Paek, Seung-Hyun Kong, and Kevin Tirta Wijaya. K-lane: Lidar lane dataset and benchmark for urban roads and highways. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE, jun 2022. URL: https://doi.org/10.1109%2Fcvprw56347.2022.00491, doi:10.1109/cvprw56347.2022.00491.

[32] Jinu Pahk, Jungseok Shim, MinHyeok Baek, Yongseob Lim, and Gyeungho Choi. Effects of sim2real image translation on lane keeping assist system in carla simulator, 2022. URL: https://arxiv.org/abs/2211.12873, doi:10.48550/ARXIV.2211.12873.

[33] Xingang Pan, Jianping Shi, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Spatial as deep: Spatial cnn for traffic scene understanding. In *AAAI Conference on Artificial Intelligence (AAAI)*, February 2018.

[34] Pedro O. Pinheiro, Tsung-Yi Lin, Ronan Collobert, and Piotr Dollár. Learning to refine object segments. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision – ECCV 2016*, pages 75–91, Cham, 2016. Springer International Publishing.

[35] Yeqiang Qian, John M. Dolan, and Ming Yang. Dlt-net: Joint detection of drivable areas, lane lines, and traffic objects. *IEEE Transactions on Intelligent Transportation Systems*, 21(11):4670–4679, 2020. doi:10.1109/TITS.2019.2943777.

[36] M. Revilloud, D. Gruyer, and MC. Rahal. A new multi-agent approach for lane detection and tracking. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3147–3153, 2016. doi:10.1109/ICRA.2016.7487482.

[37] Marc Revilloud, Dominique Gruyer, and Mohamed-Cherif Rahal. A lane marker estimation method for improving lane detection. In *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pages 289–295, 2016. doi:10.1109/ITSC.2016.7795569.

[38] Stephan R. Richter, Vibhav Vineet, Stefan Roth, and Vladlen Koltun. Playing for data: Ground truth from computer games. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *European Conference on Computer Vision (ECCV)*, volume 9906 of *LNCS*, pages 102–118. Springer International Publishing, 2016.

[39] Guodong Rong, Byung Hyun Shin, Hadi Tabatabaee, Qiang Lu, Steve Lemke, Mārtiņš Možeiko, Eric Boise, Geehoon Uhm, Mark Gerow, Shalin Mehta, Eugene Agafonov, Tae Hyung Kim, Eric Sterner, Keunhae Ushiroda, Michael Reyes, Dmitry Zelenkovsky, and Seonman Kim. Lgsvl simulator: A high fidelity simulator for autonomous driving, 2020. URL: https://arxiv.org/abs/2005.03778, doi:10.48550/ARXIV.2005.03778.

[40] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles, 2017. URL: https://arxiv.org/abs/1705.05065, doi:10.48550/ARXIV.1705.05065.

[41] Evan Shelhamer, Jonathan Long, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(4):640–651, 2017. doi:10.1109/TPAMI.2016.2572683.

[42] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2014. URL: https://arxiv.org/abs/1409.1556, doi:10.48550/ARXIV.1409.1556.

[43] Irwin Sobel. An isotropic 3x3 image gradient operator. *Presentation at Stanford A.I. Project 1968*, 02 2014.

[44] Steve Coast. Openstreetmap. URL: https://www.openstreetmap.org/.

[45] Deepak Talwar, Sachin Guruswamy, Naveen Ravipati, and Magdalini Eirinaki. Evaluating validity of synthetic data in perception tasks for autonomous vehicles. pages 73–80, 08 2020. doi:10.1109/AITEST49225.2020.00018.

[46] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. 2019. URL: https://arxiv.org/abs/1905.11946, doi:10.48550/ARXIV.1905.11946.

[47] Mingxing Tan, Ruoming Pang, and Quoc V. Le. Efficientdet: Scalable and efficient object detection. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10778–10787, 2020. doi:10.1109/CVPR42600.2020.01079.

[48] Ekin Tiu. Metrics to evaluate your semantic segmentation model, Oct 2020. URL: https://towardsdatascience.com/metrics-to-evaluate-your-semantic-segmentation-model-6bcb99639aa2.

[49] Unity Technologies. Unity. URL: https://www.https://unity.com/.

[50] Dat Vu, Bao Ngo, and Hung Phan. Hybridnets: End-to-end perception network, 2022. URL: https://arxiv.org/abs/2203.09035, doi:10.48550/ARXIV.2203.09035.

[51] Chien-Yao Wang, Hong-Yuan Mark Liao, Yueh-Hua Wu, Ping-Yang Chen, Jun-Wei Hsieh, and I-Hau Yeh. Cspnet: A new backbone that can enhance learning capability of cnn. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 1571–1580, 2020. doi:10.1109/CVPRW50498.2020.00203.

[52] Dong Wu, Manwen Liao, Weitian Zhang, Xinggang Wang, Xiang Bai, Wenqing Cheng, and Wenyu Liu. Yolop: You only look once for panoptic driving perception, 2021. URL: https://arxiv.org/abs/2108.11250, doi:10.48550/ARXIV.2108.11250.

[53] Zhenqiang Ying, Ge Li, and Guozhen Tan. An illumination-robust approach for feature-based road detection. In *2015 IEEE International Symposium on Multimedia (ISM)*, pages 278–281, 2015. doi:10.1109/ISM.2015.46.

[54] Fisher Yu, Haofeng Chen, Xin Wang, Wenqi Xian, Yingying Chen, Fangchen Liu, Vashisht Madhavan, and Trevor Darrell. Bdd100k: A diverse driving dataset for heterogeneous multitask learning. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[55] Tu Zheng, Yifei Huang, Yang Liu, Wenjian Tang, Zheng Yang, Deng Cai, and Xiaofei He. Clrnet: Cross layer refinement network for lane detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 898–907, June 2022.

[56] Xin Zheng, Qinyi Lei, Run Yao, Yifei Gong, and Qian Yin. Image segmentation based on adaptive k-means algorithm. *EURASIP Journal on Image and Video Processing*, 2018, 08 2018. doi:10.1186/s13640-018-0309-3.