# Semantic segmentation for the detection of lanes in a driving simulation environment

Omar Rodrigo Muñoz Gómez
*School of Engineering and Science*
*Tecnológico de Monterrey*
Estado de México, México
A01169881@itesm.mx

***Abstract*—Lane detection is an important part of automated driving systems, detecting the lanes allows these systems to gain a better understanding of their environment and to better navigate around them. In recent times, automotive companies have started using simulations to shorten the development time of complex models that require a lot of training and validation data. Using CARLA simulation environment and a segmentation algorithm based on UNet, we create a system that is able to detect the left and right lanes of the vehicle.**

***Index Terms*—autonomous driving, computer vision, lane detection, semantic segmentation, UNet, simulation**

## I. Introduction

In recent years there has been a rising interest in the field of advanced driving assistant systems (ADAS); companies and researchers are making great efforts to develop more intelligent vehicles that facilitate the driving task. It is envisioned that such assistants will someday provide us with a fully autonomous driving vehicle that can help us reduce the stress and dangers that driving often implies. The global autonomous vehicle market was valued at $76.13 billion in 2020 and is projected to reach $2,161.79 billion by 2030 [4]. While a fully autonomous vehicle seems like the ultimate goal, these systems can be expensive, and there are still many technical challenges ahead. However, other slightly simpler driving assistants are slowly being introduced to commercial vehicles. Some of these systems heavily rely on their understanding of the environment, particularly the location of driving lanes and can be considered the building blocks of a fully autonomous vehicle. For example, driving assistants such as Lane Departure Warning (LDW), Lane Keep Assist (LKA), and Lane Centering Assist (LCA) heavily rely on the knowledge of the lanes in the real world. Once the system is deployed into a commercial vehicle, it is expected to work under many scenarios since it might as well be used on rainy, snowy, or sunny days. While different approaches can be taken for their development, those based on supervised learning techniques often require labeled data for training the system. However, gathering the data needed to train and validate the system can be expensive. Furthermore, once the data has been gathered, it needs to be manually labeled. Because of this, companies have begun to realize that simulated environments can help them reduce costs and obtain faster development cycles [6]. In this work, we train a system for detecting the left and right driving lanes using a neural network approach based on images and labels gathered from the CARLA simulator. The trained system is then deployed to perform predictions in real-time using the same simulation environment. While a production-ready system needs to be trained under thousands of simulated environments and then validated using real-world data, we limit ourselves to training and validating our approach under controlled conditions in CARLA. It is not in the project's scope to make a production-ready lane detection system.

The document is divided into five sections, Section II introduces the simulation environment and the neural network architecture used for semantic segmentation. Section III describes the methodology and the main results obtained in this work. Section IV discusses our results. Finally, Section V presents the conclusions and future work.

## II. Background

### A. CARLA simulation environment

Carla is an open-source simulator developed at the Computer Vision Center (Barcelona, Spain) for autonomous driving research. It supports the development, training, and validation of autonomous driving systems. It features a C++ and Python API to control all simulation aspects: traffic, pedestrians, weather, and sensors, among others. It also provides an autonomous driving sensor suite, including LIDAR, cameras, depth sensors, and GPS. Additionally, it can generate user-defined maps and perform traffic simulations [1].

### B. UNet

UNet is a symmetric U-shaped fully convolutional neural network initially developed for biomedical image segmentation [5]. UNet is based on an encoder-decoder architecture. It uses skip connections to combine high-level semantic feature maps from the decoder and corresponding low-level detailed feature maps from the encoder [2]. In this work, we use a modified UNet based on [3], which works with gray-scale images at its input, and it outputs a three-channel segmentation map corresponding to the three possible classes in our problem: background, left lane, and right lane. The architecture is shown in Figure 1.

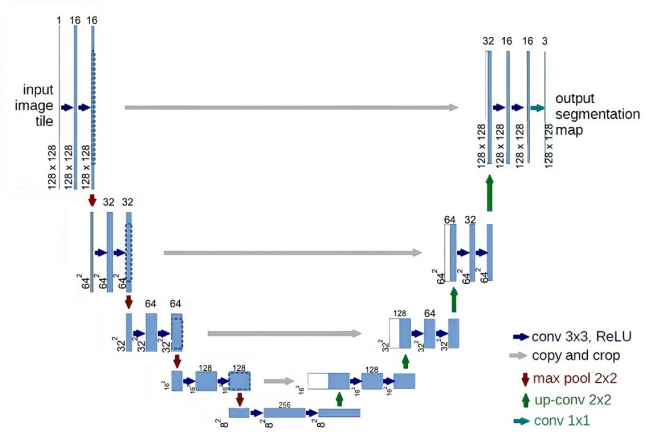



Fig. 1: UNet architecture used for semantic segmentation of driving lanes. The output segmentation map contains three channels corresponding to the background, the left, and the right lanes.

## III. Methodology and Results

Our objective is to run the simulation environment and perform real-time lane detections using our trained system. We divide our methodology in two main stages: model training, and model deployment. These two stages are summarized in Figure 2. The most important parts of this methodology are discussed in the subsections below.

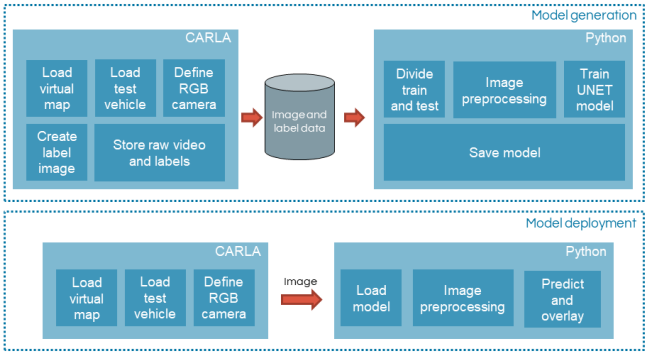



Fig. 2: Two staged methodology for performing real-time lane detections using CARLA simulator. The first stage consists of extracting the data (raw image and labels), and training the UNet weights. The second stage consists of deploying the model to perform real-time predictions.

### A. Data set creation

For generating our data set, we spawned a single vehicle into a virtual world in CARLA and defined an RGB camera in the front of the vehicle. As the vehicle travels through the virtual world, the raw video image is saved and the label image is first computed and then saved.

*1) Raw camera image:* We define a virtual RGB camera attached to the vehicle's frame of reference with an offset of 0.5 meters in the X direction and 1.3 meters in the Z direction, this corresponds to a front-facing camera. The camera has a resolution of 1024x512 pixels, with a field of view of 45°, a pitch angle of 5°, a roll angle of 0° and a yaw angle of 0°.

*2) Label image:* While CARLA contains a semantic segmentation sensor that returns the labels of the objects in the map, the segmentation image does not contain the information that we want in the format that we want. The segmentation map is able to distinguish between 13 possible classes, including buildings, pedestrians, vehicles, road lines, traffic, signs, among others. However, for road lines it is not returning the left and right lanes with respect to the driving vehicle, instead it returns all possible road lines. To get the labeled image, we then make use of the world coordinates ($X_w$, $Y_w$, $Z_w$) of the predefined path that the vehicle is travelling. The world coordinates of the path can be converted into an image by converting the world coordinates into camera coordinates, and then into positions in the image. This transformation is described in terms of the instrinsic (K) and extrinsic (R|t) camera matrices expressed in homogeneous coordinates in Equation 1. Only the first 60 points of the vehicle's path are used for creating the label image. The path is a line along the center of the lane and to compute the information for both lanes, we simply apply an offset of half of the lane width to each side of the path coordinates.

$$\lambda \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \mathbf{K}\ (\mathbf{R}|\mathbf{t}) \begin{pmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{pmatrix} \tag{1}$$

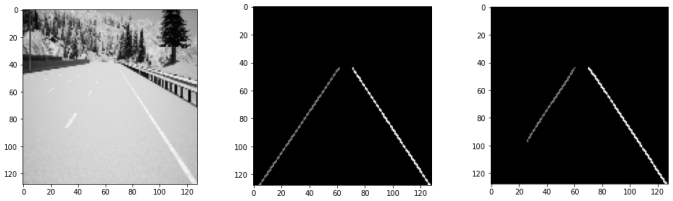
### B. Model training

We ran the data set creation script until around 500 images were obtained. We use this image data set for training and validating the neural-network model (UNet). The data set was divided into training (75%) and testing (25%) splits. Since the project is not meant to develop a production-ready system, this amount of data allows us to quickly assess the changes to our model and to get a good idea of the proof-of-concept performance.

Before training the UNet model, we perform transformations to the raw camera frames and labels. For a faster training and prediction time, we resized the raw camera frames to 128x128, converted the RGB image to grayscale, and normalized it in the 0 to 1 range. Additionally, the label image is resized to 128x128 using inter-nearest interpolation (which avoids undesired artifacts in the labels caused by the resizing). Finally, we transform each labeled image to a three-channel image, which is the format expected by the UNet during training (each channel corresponds to one class), and train the model for 50 epochs using a batch size of 16 and categorical cross-entropy as the loss function. The total number of trainable parameters of our model is 1,940,851.

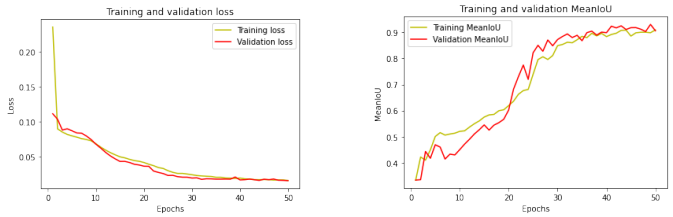
### C. Performance evaluation

Throughout the training, the performance is tracked for both the training and test sets (validation). Figure 4a shows the effect of training in the loss function and additionally we kept track of the mean intersection over union (Mean IoU) which is another metric typically used for evaluating the performance for segmentation algorithms, the mean intersection over union for the different epochs is shown in Figure 4b. For a visual evaluation of the performance, Figure 3 shows one sample image from the testing data set, together with the ground truth labels and the predicted labels.



(a) (b) (c)

Fig. 3: Visual assesment of the performance of the model against the testing data. Image (a) shows the input image which is the result of the preprocessing stage, image (b) shows the ground truth label and (c) shows the predicted labels. We observe that the left lane prediction has missing sections with respect to the ground truth labels.




(a) (b)

Fig. 4: Evolution of the performance indicators throughout the training process for the training and validation sets. Image (a) shows the categorical cross-entropy loss function and image (b) shows the Mean IoU.

The resulting model is 22.4 MB in size and it takes roughly 50 ms on average to make a single prediction. This implies we can make predictions at around 20 FPS which is close to "real-time".

### D. Model deployment

Once the model has been trained, we deploy the model to make predictions in real-time. The model deployment process is described in Figure 2 and it consists of loading the simulator environment and defining a virtual camera. Then, we obtain the camera image using a Python script and perform a preprocessing stage to the video frames to meet the same conditions as the images used to train the UNet model, that is, an image resizing and normalization is performed. Finally, we load the pre-trained model and feed the preprocessed images to obtain predictions as the vehicle moves around the virtual world. Once a model has been deployed in a real application, we often don't have access to labeled data and the performance needs to be measured either visually or based on the complete system performance, e.g., the performance of a lane centering function which uses as one of its subsystems a lane detection algorithm. Figure 5 shows two different frames of the predictions made by the model during its deployment on images never seen before by the algorithm. The predicted lanes are overlaid in the input image; the predicted left lane is shown in blue, while the predicted right lane is shown in red. We observe that there are multiple fake detections of the lanes. However, the actual lanes are also detected for the most part.

(a) (b)

Fig. 5: Two different example predictions of the deployed model on unseen images running in "real-time". We have both false negative and false positive predictions. However, the algorithm is able to detect the actual lanes to some extent.

## IV. Discussion

### A. Performance on unseen images

Once a model has been deployed in a real-system, we often don't have access to the labeled data and we need to be able to assess the performance somehow else. One possible way to do this is to ask users for feedback of the system as a whole. Additionally, during development tests, we can also define different KPIs for the system. For example, for the lane departure warning, the system is evaluated on its capability to alert the user when the vehicle is actually leaving the lane and we attempt to reduce the false alerts. During the development, data is collected and whenever a false alert is detected, this data can be sent back to developers for analysis. The root cause of the problem may be tracked down to different components of the system and in case the lane detection algorithm is found responsible for the false alert, the system may be trained back using additional data to try to mitigate the poor lane detections. We observe that our system is making poor lane detections in the examples from Figure 5. However, considering the amount of data that was used during training, this is expected. The data set needs to be expanded and techniques such as data augmentation could also help us improve the performance of our model.

### B. Real-world predictions

Some of the advantages of training a model using a simulation environment include the rapid development cycles, the

simulation of corner cases, and the reduced costs of collecting data which cover the acquisition of testing vehicles, the data collection team, the associated equipment, and the labeling team, among others. However, we want to translate our models to work in real-world vehicles. We need to make additional changes to our proposed model to achieve this objective. For example, in the previous section, we briefly discussed the need for a much larger data set and techniques such as data augmentation to create a more robust model against different unseen scenarios. However, perhaps more important is that the system is trained using a simulation environment and is meant to be used in a real-world vehicle. We need to fine-tune the model to work with real-world images, as the deep-learning models can be pretty sensitive to the input data used to build the model. In this sense, we could train our system in simulation with thousands of simulated environments and tests and finally fine-tune the algorithm with a more minor data set of real-world images and labeled data. To make our model more robust, we could train it in higher-quality simulation environments- environments where the images are photo-realistic.

## V. Conclusions and future work

In this work, we developed a semantic segmentation algorithm for detecting the driving lanes in a driving simulation environment based on the UNet architecture. The model was trained and validated using a self-made data set within the simulation environment. After 50 epochs, we obtain a loss of less than 0.05 and a Mean IoU greater than 0.9 on the testing data. While the loss and metrics used during testing have very good results, the data set used is small (around 500 images total). Once the model is deployed, it can perform predictions at around 20 FPS on average, which is close to real-time. However, the model may be further modified to run faster by performing experiments to determine the minimum image size needed for accurate lane detections. When the algorithm receives previously unseen video frames, we observe clear false positive and false negative predictions. However, considering the size of the training data set, this performance was expected. We need a more extensive training and validation set to obtain a model that can generalize to many scenarios. The advantage of using a simulation environment is that such scenarios can be synthetically generated. Additionally, we may obtain better performance by performing data augmentation. Furthermore, since the system is trained and validated using simulated data, other techniques need to be used to be able to take this system to a real-world vehicle. For instance, we propose to perform a fine-tuning of the model based on real-world data, always to test the performance against real-world data, and to enhance the simulation environment as much as possible to use photo-realistic images during training.

## References


[1] CARLA Team. CARLA. https://carla.org/, 2022. Accessed: 2022-11-27.

[2] Huimin Huang, Lanfen Lin, Ruofeng Tong, Hongjie Hu, Qiaowei Zhang, Yutaro Iwamoto, Xianhua Han, Yen-Wei Chen, and Jian Wu. Unet 3+: A full-scale connected unet for medical image segmentation, 2020.

[3] Raoof Naushad. UNET for Semantic Segmentation - Implementation from Scratch. https://medium.datadriveninvestor.com/unet-for-semantic-segmentation-implementation-from-scratch-26e043fdcffa, 2021. Accessed: 2022-11-27.

[4] Research and Markets. Autonomous Vehicle Market by Level of Automation, Application, Drive Type, and Vehicle Type: Global Opportunity Analysis and Industry Forecast, 2021-2030. https://www.researchandmarkets.com/reports/5578161/autonomous-vehicle-market-by-level-of-automation, March 2022. Accessed: 2022-11-21.

[5] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015. cite arxiv:1505.04597Comment: conditionally accepted at MICCAI 2015.

[6] Tesla. Tesla AI Day 2021. https://www.youtube.com/watch?v=j0z4FweCy4M. Accessed: 2022-11-27.