Lane detection in miniature environments using image segmentation

Markus Kasten[1]

HAW-Hamburg, Dept. for Computer Science

Abstract. Lane detection is an important task when it comes to developing advanced driver assistance systems (ADAS) and autonomous vehicles. While much research is dedicated to real world vehicles, we want to try a machine learning based lane detection approach in a 1:87 miniature environment. With a fixed network architecture, we experiment with different training methods to learn how both existing real world data and specialized training data for our environment affects lane detection performance. While only using real world training data granted usable results, using a low number of specialized training images greatly improved performance. Yet, further experiments are necessary for reliable lane detection.

Keywords: Machine Learning · Image Segmentation · Autonomous Driving · Lane detection · Miniature Autonomy

1 Introduction

Detecting driving lanes in a road environment is one of the most important aspects of autonomous driving. Yet, while easily distinguishable for humans, finding lanes in varying conditions using traditional image processing algorithms is not an easy task. Using machine learning, we aim to provide a robust method that can either assist algorithmic lane detection or fully replace it.

Since research on full size vehicles on public roads comes with significant risks, we are using a 1:87 scale (H0-scale) environment for our tests. This involves a miniature scale world, shown in Figure 1, has been purpose built to test autonomous driving scenarios[5], as well as a scale vehicle equipped with a single front facing camera and various sensors[6].

2 Related work

Lane detection is a very actively researched field. While traditional image processing methods have been researched a long time ago[13][14], recent developments have shifted towards deep learning based methods. The following chapter describes some of these methods and related works that have been created based on them.

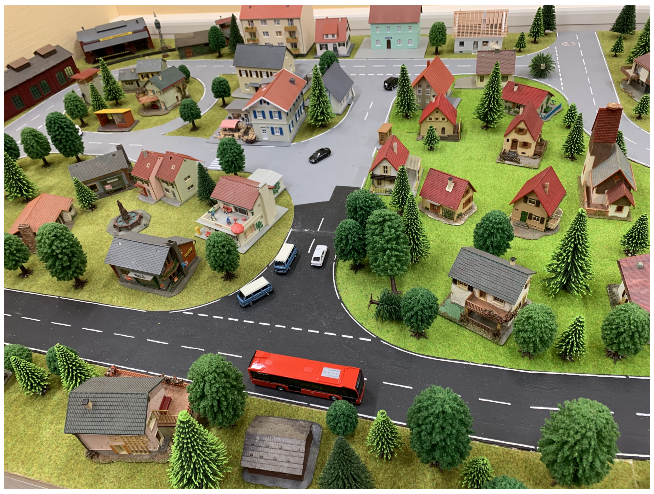


Fig. 1. Miniature enviroment designed to test autonomous driving methods

3 Methods

The methods presented and used in this paper purely rely on color images from the vehicles front facing camera. While other approaches also use LIDAR [1] [2] or depth-sensing using stereo-cameras[8], this paper does not use these additional data-sources as they are not available on our H0-scale vehicles as of now.

For extracting lane information from an image, we need to define a output data representation. The output data format should be easy to work with in further processing tasks, but also allow for complex lane shapes.

One representation is a set of polynomials in the form of $p(x) = a_0 + a_1x + a_2x^2 + ... + a_nx^n$ using n coefficients, each of which is determined by a neural network. It would allow very easy processing in e. g. lane keeping, but with a fixed number of output variables and therefore fixed number of polynomials, the complexity of detected lanes is limited. Representing intersections or turning lanes at the same time with driving lanes can be error prone. This method was used among others in PolyLaneNet [12] with a regression based approach.

Another output representation is to create an image of the same dimensions as the input image, which has the lanes rendered as easy distinguishable lines. Such an output format can be achieved by using a fully convolutional neural network (FCNN) [7]. This approach is also called image segmentation, as the image is segmented into semantic elements. If needed, spline data can be easily extracted using conventional image processing methods. This method has become very popular in recent years and is still very actively researched. Some examples include SCNN[15] or ENet-SAD[4].

In this paper we focus on FCNNs, since this is the most common approach and allows for more errors. We want to compare how the network performs in respect to different types of training data, number of epochs and network parameters.

3.1 Selecting a network architecture

With the goal of using a Google EdgeTPU in the future, we are limited to using TensorFlow 2. This greatly reduces the availability of existing FCNN implementations. For it's performance and speed, an ERFNet architecture[10] has been selected and will be used throughout this paper. It only has $1,988,972$ trainable parameters, allowing fast training and inference.

This type of neural network consists of a encoder, also called downsampler, and a decoder, also called upsampler. The encoder block uses a sequence of convolution layers to process the input image, the decoder block uses a set of deconvolution layers to generate the output image. In this paper we refer to the output image of the network as its prediction.

We are using the Mean Squared Error function for calculating loss and the Nadam optimizer with a learning rate of 0.001.

3.2 Labeling

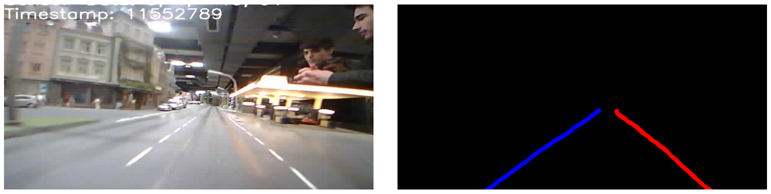
We want our network to draw a line along the left and right limits of the ego-lane. Other lanes such as intersections, parallel and oncoming lanes are ignored. The limits of the ego-lane are usually defined by lane markers on both sides, but one or both markers may be missing. Labels are drawn up to the vanishing point, or to a point where lines are no longer separable. All lines are drawn with a width of 10 pixels.

Three-channel images are used for the input and output of the network. Input images are presented as an RGB image. Other color-spaces such as LAB or YUV have been considered, but research has shown that performance gains between color-spaces are negligible[3]. The output (prediction) uses the blue channel for identifying the left lane limit and the red channel for the right lane limit. The green channel is unused and always black, but could be used for identifying other features in the future.

3.3 Gathering training data

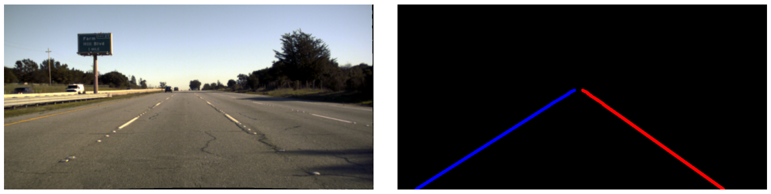
Training datasets of annotated images for lane detection are available[2][15], but aren't designed to be used in 1:87-scale environments. Example image from the LLAMAS and Culane datasets are shown in Figure 3 and 4 respectively, an example image from our 1:87 vehicle is shown in Figure 2.

As part of the process, a total of 96 images have been hand-annotated, 53 of which are used as training and validation data, 43 are used as test data. These images have been annotated by drawing the lane markers by hand in a image editor and therefore aren't as precise as the automatic annotations from the LLAMAS dataset.



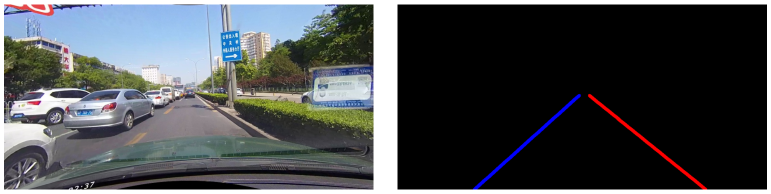
(a) Input (b) Mask

Fig. 2. Example image taken from the front facing camera of a vehicle in a 1:87 environment (Miniatur Wunderland Hamburg)



(a) Input (b) Mask

Fig. 3. Random image taken from the LLAMAS dataset. These images are mostly from a highway environment.



(a) Input (b) Mask

Fig. 4. Random image taken from the Culane dataset. These images are mostly from a city environment with occlusions from the ego-vehicle, and very dense traffic.

All images and masks are scaled to a resolution of 1024×512 pixels, making a compromise of preserved detail in images, and training and inference performance. Dimensions of 2^n were chosen to avoid rounding errors in pooling layers resulting in a different output resolution from the input resolution.

To increase the amount of data, all training and validation data are passed through an image augmentation pipeline. Data augmentation is an effective method for avoiding overfitting and increasing network performance[11]. This pipeline varies brightness, contrast, hue and saturation of all images by a random amount. These augmented images are then combined with the original data, effectively doubling the dataset in size.

3.4 Experiments

We want to compare a number of different training methods for our segmentation network. The experiments mostly differentiate in the combination of datasets used as training data, but also vary in the total number of training images, epochs and whether data augmentation is used. Experiments are listed in Table 1.

The number of training epochs has been chosen experimentally by monitoring the loss graph. For training with LLAMAS and Culane data, 50 epochs have been chosen as there aren't any gains beyond this epoch (see Figure 5). For training with our own very small dataset, 120 epochs are needed as shown in Figure 6.

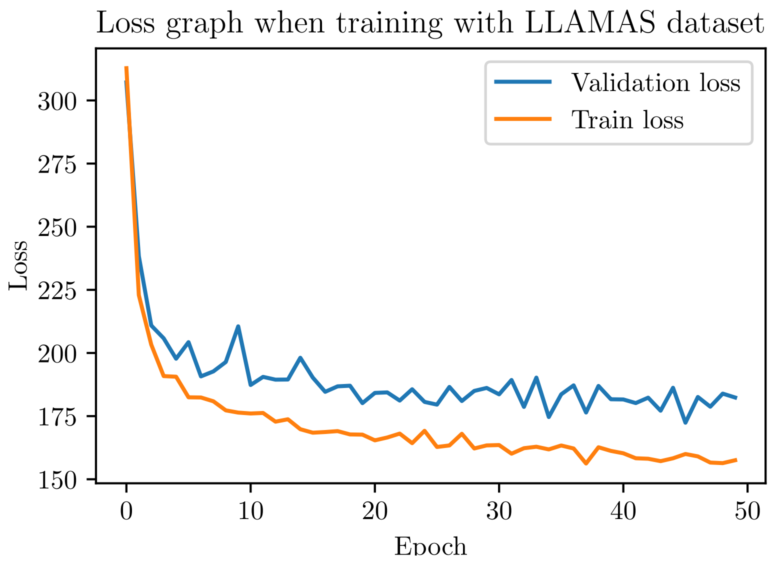


Fig. 5. Plot of loss vs. epoch when training with the LLAMAS dataset

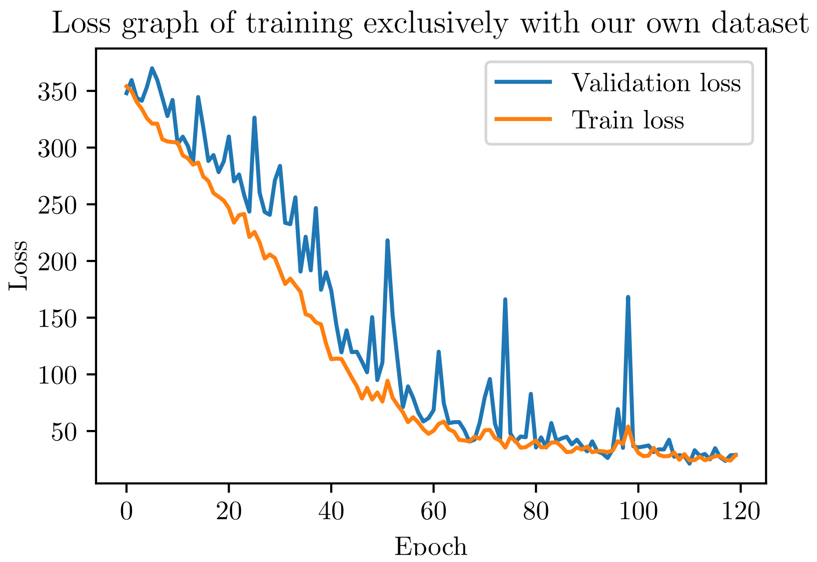


Fig. 6. Plot of loss vs. epoch when training with our own dataset

Each network is trained on a single NVIDIA Tesla V-100. The network is then tested against the 43 images of our self-annotated test dataset.

Table 1. Table of training methods we are evaluating with our network

Datasets	Total training images	Augmentation	Epochs	Learning rate
LLAMAS	1000	Yes	50	0.001
LLAMAS	1000	No	50	0.001
LLAMAS+Culane	2000	Yes	50	0.001
LLAMAS+Culane	2000	No	50	0.001
Our dataset	53	Yes	120	0.001
Our dataset	53	No	120	0.001
LLAMAS+Culane+Ours	2053	Yes	50	0.001
LLAMAS+Culane+Ours	2053	No	50	0.001

4 Results

In this section we discuss results gathered by different methods of training the neural network, and how we compare network performance.

4.1 Evaluation metrics

For comparing network performance, first the predicted masks by the neural network is binarized, i. e. a pixel is either True or False. A threshold of 0.5 has been chosen for all tests. Three performance metrics have been selected, first of which is the intersect over union (IoU, also called Jaccard-Coefficent) for measuring the overlap of the predicted mask P and the ground truth T.

$$IoU = \frac{|P \cap T|}{|P \cup T|}$$

We also compare the precision P and recall R using pixel-wise true positives TP, true negatives TN, false positives FP and false negatives FN. Precision tells us how many pixels had been correctly identified as a lane limit over all detected pixels.

$$P = \frac{TP}{TP + FP}$$

Recall describes the completeness of the predicted mask, comparing correctly predictions against missing predictions.

$$R = \frac{TP}{TP + FN}$$

For calculating each of the metrics of our defined training methods, all metrics are calculated for every test image in our dataset, and then averaged over all images.

4.2 Experiment results

The presented metrics have been calculated for all training methods. The results are presented in Table 1 and plotted in Figure 7. The predictions of our network are shown in Figure 8.

Table 2. Results for the each of the tested training methods. Best overall results are highlighted.

Datasets	IoU	Precision	Recall
LLAMAS Aug.	0.08249	0.08438	0.76453
LLAMAS	0.06047	0.06131	0.81897
LLAMAS+Culane Aug.	0.09159	0.09306	0.84416
LLAMAS+Culane	0.07182	0.07284	0.83067
LLAMAS+Culane+Own Aug.	0.11809	0.11969	0.87758
LLAMAS+Culane+Own	0.08103	0.08149	**0.92407**
Own Aug.	**0.31011**	**0.41009**	0.55226
Own	0.20347	0.23635	0.57879

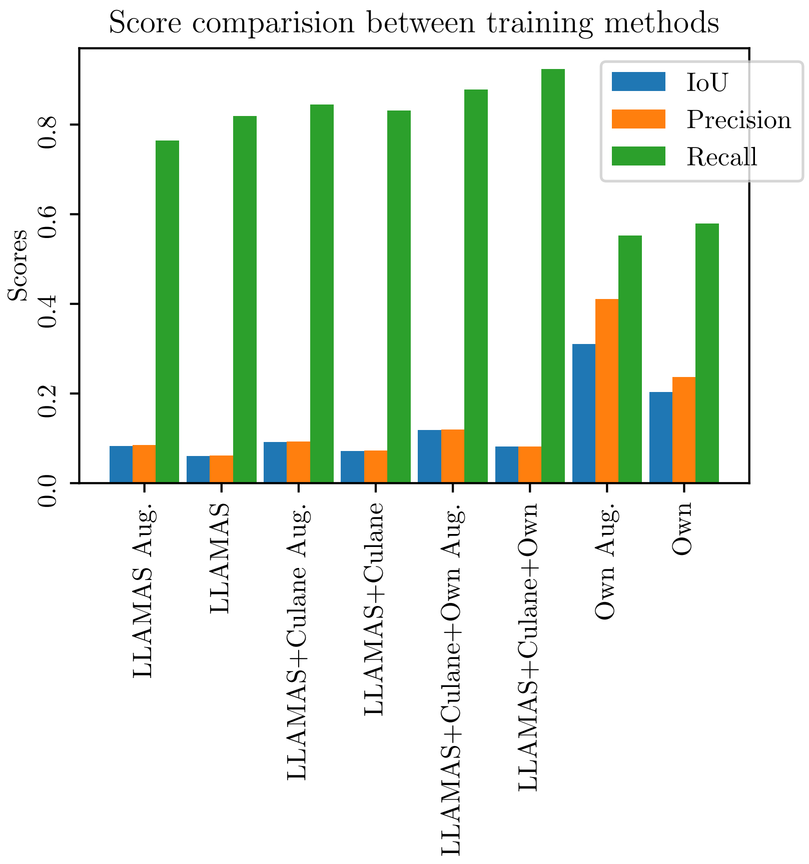


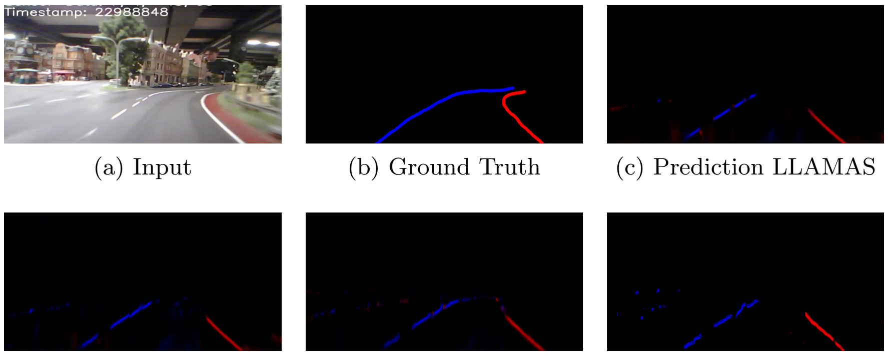
Fig. 7. Scores of different training methods plotted

When comparing training with and without augmented data, we can clearly see an improvement when using augmented data. This result was expected as data augmentation increases robustness and avoids overfitting to the given training data.

A slight improvement can be observed when Culane data was mixed with LLAMAS data. This is expected as well, since additional data widens the variety of inputs and further avoids overfitting. Additionally, Culane data is mostly from city environments, which is more similar to the test data from the miniature environment.

Adding annotated images from the miniature environment to the training data improves results even more, despite only being a very small fraction of training data (one image on 20 or 40 LLAMAS/Culane images).

Unexpected results were observed when training exclusively with our very small dataset. While IoU and Precision metrics have improved by a factor of over 3, Recall has decreased by nearly 40%.



(a) Input (b) Ground Truth (c) Prediction LLAMAS

(d) Prediction LLAMAS+Culane (e) Prediction LLAMAS+Culane+Ours (f) Prediction Ours

Fig. 8. Input and ground truth, with the prediction of each of our trained networks. All shown networks are using augmented data.

4.3 Comparison with other research

LLAMAS has trained a DeepLabV3+ achieving an average precision of 31.2% when tested against their test dataset. This value is higher than any of our tests, excluding using just our own training data, which is expected when testing against test data from another dataset.

5 Conclusion

Our experiments have shown that purely relying on real world data for training a network to identify lanes in 1:87 environment does not unlock the networks full potential. Using data augmentation and adding even small amounts data from a miniature environment improves results notably. We have also shown that usable results can be achieved when only using a very small dataset of the same environment, but these are likely the cause of overfitting and need further testing and investigation.

To use the predictions of our network in an autonomous driving scenario, further image processing is needed. A modified algorithm for lane detection used in environments with easily distinguishable markers such as the Carolo-Cup could be used, but this has not been tested so far.

6 Prospects

While the gathered results are satisfying, they are barely usable for reliable autonomous driving in the used miniature environment. Instead of extracting lanes from a single image, a series of images could be used to improve confidence where lane markers are either occluded or not present.

Additionally, more advanced neural network architectures such as [9] or [12] should be evaluated in the miniature environment.

Inference on a Google EdgeTPU has not been achieved as part of this paper due to compatibility problems, but is still an option for future developments.

7 Acknowledgments

We thank Stephan Pareigis for introducing us to this field of research, and the HAW Hamburg ITSC for providing significant computing resources, allowing our comparisons in acceptable time frames. We also thank Luk Schwalb for the useful discussions and suggestions on this paper.

Bibliography

[1] Min Bai, Gellért Máttyus, Namdar Homayounfar, Shenlong Wang, Shrinidhi Kowshika Lakshmikanth, and Raquel Urtasun. Deep multi-sensor lane detection. *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3102–3109, 2018. 3

[2] Karsten Behrendt and Ryan Soussan. Unsupervised labeled lane marker dataset generation using maps. In *Proceedings of the IEEE International Conference on Computer Vision*, 2019. 3, 3.3

[3] Shreyank N. Gowda and Chun Yuan. Colornet: Investigating the importance of color spaces for image classification. In *ACCV*, 2018. 3.2

[4] Yuenan Hou, Zheng Ma, Chunxiao Liu, and Chen Change Loy. Learning lightweight lane detection cnns by self attention distillation. *arXiv preprint arXiv:1908.00821*, 2019. 3

[5] Markus Kasten, Luk Schwalb, and Morten Stehr. H0-scale street platform, June 2020. 1

[6] Markus Kasten, Luk Schwalb, and Morten Stehr. Miniaturautonomie im H0-Maßstab, 2020. 1

[7] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation, 2014. 3

[8] A. Meyer, N. O. Salscheider, P. F. Orzechowski, and C. Stiller. Deep semantic lane segmentation for mapless driving. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 869–875, 2018. 3

[9] Zequn Qin, Huanyu Wang, and Xiao jie Li. Ultra fast structure-aware deep lane detection. *ArXiv*, abs/2004.11757, 2020. 6

[10] E. Romera, J. M. Álvarez, L. M. Bergasa, and R. Arroyo. Erfnet: Efficient residual factorized convnet for real-time semantic segmentation. *IEEE Transactions on Intelligent Transportation Systems*, 19(1):263–272, 2018. 3.1

[11] Connor Shorten and Taghi M. Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):60, Jul 2019. 3.3

[12] Lucas Tabelini Torres, Rodrigo Ferreira Berriel, Thiago M. Paixão, Claudine Badue, Alberto Ferreira de Souza, and Thiago Oliveira-Santos. Polylanenet: Lane estimation via deep polynomial regression. *ArXiv*, abs/2004.10924, 2020. 3, 6

[13] Tsung-Ying Sun, Shang-Jeng Tsai, and V. Chan. Hsi color model based lane-marking detection. In *2006 IEEE Intelligent Transportation Systems Conference*, pages 1168–1172, 2006. 2

[14] Yue Wang, Dinggang Shen, and Eam Khwang Teoh. Lane detection using spline model. *Pattern Recogn. Lett.*, 21(9):677–689, July 2000. 2

[15] Ping Luo Xiaogang Wang Xingang Pan, Jianping Shi and Xiaoou Tang. Spatial as deep: Spatial cnn for traffic scene understanding. In *AAAI Conference on Artificial Intelligence (AAAI)*, February 2018. 3, 3.3