MASTER THESIS

Lane Detection based on Multiple Frames Information

Author:
Diego TURRADO BLANCO

Supervisor:
Dr. Ján KOLODA
Dr. Mariano RINCÓN ZAMORANO

Master in Advanced Artificial Intelligence

Higher Technical School of Computer Engineering

National University of Distance Education (UNED)

September, 2021

Declaration of Authorship

I, Diego TURRADO BLANCO, declare that this thesis titled, “Lane Detection based on Multiple Frames Information” and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

Acknowledgements

Firstly, I want to thank my thesis advisor Dr. Ján Koloda and Dr. Mariano Rincón for their advice, contribution and support in the development of this thesis. I also wish to acknowledge the help of AVL Software and Functions, providing the necessary equipment to develop this work. And last, but not least, I would like to thank my family for their support during all my studies.

NATIONAL UNIVERSITY OF DISTANCE EDUCATION (UNED)

Abstract

Master Thesis

Lane Detection based on Multiple Frames Information

by Diego TURRADO BLANCO

Keywords: lane detection, autonomous driving, convolutional neuronal networks, recurrent neuronal networks

One of the fundamental challenges in the field of autonomous driving is the ability to detect dynamic objects, such as vehicles or pedestrians, and statics ones, such as lanes, in the surroundings of the vehicle. The accurate perception of the environment is crucial for a safe decision making and motion planning. In recent years, advanced driver-assistance systems (ADAS) are getting more important, incorporating new features that a few years ago were limited to luxury cars or even not technically possible. In particular, one of this features is the lane keeping assist system which keeps the car centered in the lane. This system is not just a relevant part of actual driving support features, but it is a crucial function of future fully autonomous vehicles (AD). A few years ago, many of these systems relied on traditional computer vision algorithms, based on computational expensive manually calibrated methods. Their lack of robustness under the long tale of driving scenarios makes them not very suitable for scalability. Moreover the limited computational resources of embedded system puts additional requirements on the design of real time capable algorithms.

Nowadays, the state-of-the-art object and structure detectors for advanced driver-assistance systems are based on machine learning and, in particular, deep learning approaches. However, such approaches still mainly function on single-frame basis and do not exploit the (high) temporal correlation of the signals representing the perceived environment. Single-frame detection networks might work well under circumstances where the lanes are perfectly visible, but show a lack of performance under certain situations, like occlusions, shadows, rain, snow, lane degradation, etc. To address the aforementioned problem, this thesis introduces temporal information for lane binary segmentation, applying convolutional long short-term memory (ConvLSTM) and convolutional neural networks (CNN) to improve substantially the performance of single-frame architecture under challenging and adverse situations.

Contents

Declaration of Authorship ii

Acknowledgements iii

Abstract iv

1 Introduction 1
- 1.1 Motivation . . . 2
- 1.2 Problem statement . . . 2
- 1.3 Hypothesis . . . 3
- 1.4 Objectives and methodology . . . 3
- 1.5 Document structure . . . 3

2 Literature review and related work 5
- 2.1 Classical computer vision methods . . . 5
- 2.2 Deep learning . . . 7

3 Proposed method and implementation 9
- 3.1 Basic network architecture . . . 9
- 3.2 Encoder/Decoder selection . . . 11
- 3.3 Variations . . . 12

4 Experimental results 14
- 4.1 Dataset . . . 14
- 4.2 Implementation . . . 16
- 4.3 Metrics . . . 16
- 4.4 Training . . . 17
- 4.5 Performance . . . 20
- 4.6 When is the network really over-fitting? . . . 21
- 4.7 Comparison with state-of-the-art . . . 23
  - 4.7.1 Quantitative performance: pixel level . . . 23
  - 4.7.2 Quantitative performance: lane level . . . 25
  - 4.7.3 Qualitative performance . . . 27
  - 4.7.4 Running time . . . 29

5 Discussion 30

6 Conclusion **32**

Bibliography **33**

Appendix A Student's t-test **37**

Appendix B Challenging conditions on the TuSimple dataset **39**

List of Figures

1.1 Lane detection using LaneNet [43] under driving scenarios with shadows, light reflections and lane occlusions from the TuSimple dataset. Green lanes represent the ground truth and red, the one inferred by the CNN. . . 2

2.1 Encoder/decoder architecture. Left UNet and right SegNet. For simplicity reason batch normalization and activation layers after convolutional layers are not shown. 7

3.1 Network architecture for the special case of 5 frames and ConvLSTM2D with 2 layers. The basic idea of this architecture for lane detection was proposed in [59], however the encoder/decoder can be adapted and further changes can be done to improve not just performance but also other aspects, like running time or size. 10

3.2 Number of trainable parameters for the different architectures (encoder and decoder variations) used during this work. It is just an indicative due to the fact that small changes in the LaneNet-LSTM have been done to try to improve its performance. As basic for this figure, a 2-layer and 32-filter ConvLSTM is used. Another take away from this figure is that ConvLSTM represents 39.3% of the trainable parameters in the LaneNet-LSTM, 71.6% in the UNet-LSTM and 56.2% in the case of SegNet-LSTM. 12

3.3 Loss and validation loss for LaneNet-LSTM. Fluctuations on the validation loss could indicate over-fitting problems. To try to mitigate this changes in the network size, regularization and batch size have been proposed. . . . 13

4.1 Example of TuSimple dataset [2]. Even distributed horizontal lines are plotted in red and the intersection with each polyline with a green point. . 15

4.2 Ground truth lane is represented with the color white. Lanes with color green and red are two possible predictions with the same binary cross entropy error, although the green one has an smaller "error". 18

4.3 Training and validation loss for the different proposed loss functions. Soft binary cross entropy does not show any effect on convergence speed. . . . 19

4.4 During the training of some of the networks we have seen the following phenomena: validation loss, validation accuracy, validation recall and validation precision increase simultaneously. Intuitively the relationship between validation loss and the other three metrics should be inverse. 22

4.5 Box plots for area under ROC (receiver operating characteristic curve), under PRC (precision recall curve), maximum accuracy and its corresponding F1 for the compared state-of-the-art networks. 24
4.6 Three examples of images in challenging situations. Top row: LaneNet. Middle row: LaneNet-LSTM. Botton row: UNet-LSTM. Green lanes correspond to ground truth and red lanes to predictions (yellow is just an overlap of both of them). 25
4.7 Example result of the Hough transform applied to the segmentation map. 26
4.8 Example result for region growing. Top: raw image. Middle: output of the clustering. Bottom: result after line fitting. 26
4.9 The 20 predicted scenarios with the worst AUPRC calculated by LaneNet. 28
4.10 The 20 predicted scenarios with the worst AUPRC calculated by LaneNet-LSTM. 28
4.11 The 20 predicted scenarios with the worst AUPRC calculated by UNet-LSTM. 28

B.1 Some of the frames categorized as challenging in the TuSimple validation dataset. 39

List of Tables

4.1 Characteristic of TuSimple dataset [2]. * just the frame 20th is labeled. . . . 14
4.2 Influence of the different loss functions on the area under ROC (receiver operating characteristic curve), under PRC (precision recall curve), maximum accuracy and its corresponding F1. 19
4.3 Area under ROC (receiver operating characteristic curve), area under PRC (precision recall curve), maximum accuracy and its corresponding F1 for the different combinations of LaneNet-LSTM. The results correspond to the better results obtain for each architecture. 21
4.4 Area under ROC (receiver operating characteristic curve), under PRC (precision recall curve), maximum accuracy and its corresponding F1 for the compared state-of-the-art networks. * refers to LaneNet-LSTM with 32 filters ConvLSTM2D and dropout of 0.1. 23
4.5 Area under ROC (receiver operating characteristic curve), under PRC (precision recall curve), maximum accuracy and its corresponding F1 for the compared state-of-the-art networks for only the challenging situations contained in the TuSimple validation set. * refers to LaneNet-LSTM with 32 filters ConvLSTM2D and dropout of 0.1. 24
4.6 Average inference time. * refers to LaneNet-LSTM with 32 filters ConvLSTM2D and dropout of 0.1. 29

A.1 Results of Student's t-test for the comparison between LaneNet, LaneNet-LSTM and UNet-LSTM. 38

Chapter 1

Introduction

In recent years, the development of computer vision (i.e. new deep learning architectures), sensor technology (i.e. LIDAR) and processors units (i.e. GPU) has made big advances in the field of autonomous driving possible. If fact, Scopus database confirms that it has become a prominent topic of investigation for researchers, not just in the academia but also in the industry [42], where great amount of effort and money has been invested in. The ultimate goal of many of these researches is to have a full picture of the environment around the vehicle, detecting dynamic objects such as vehicles or pedestrians, and statics ones such as lanes, and having a semantic understanding of the surroundings which allows a proper movement planning. One of the key and at the same time basic features needed to enable autonomous driving is camera based lane detection. Once the lanes are detected, the vehicle can have a reference to position itself in the surrounding world, so the trajectory planning can calculate where to go, reducing the possibility of collision with vehicles driving in other lanes [53]. Therefore designing a robust camera based real time capable lane detection is a key milestone for AD/ADAS systems.

Different approaches have been explored for lane detection. They can be classified into two different groups: traditional computer vision algorithms and, specially in recent years, deep learning based methods. Theses different approaches will be presented and investigated in further detail during chapter 2.

Although the architecture of both approaches is totally different, most of these systems are based on the information contained in one single frame, and just a few of them take advantage of the information contained in previous frames [59]. This, of course, is not like human drivers work, being able to extrapolate and inference the position of the lanes under challenging situations, like shadows, light reflections, lane occlusion, etc. referring to the information of the past. This can be done because lanes are static, continuous objects on the street, with a huge overlapping between frames, meaning a highly related and temporal correlation of the signals. So the lanes in the actual frame could be partially inferred from the information in the n-last frames, even though the lanes might not be totally visible anymore.

Under those circumstances the performance of the state of-the-art methods decreases, detecting the lanes erroneously, in another direction, or even not detecting it at all like in

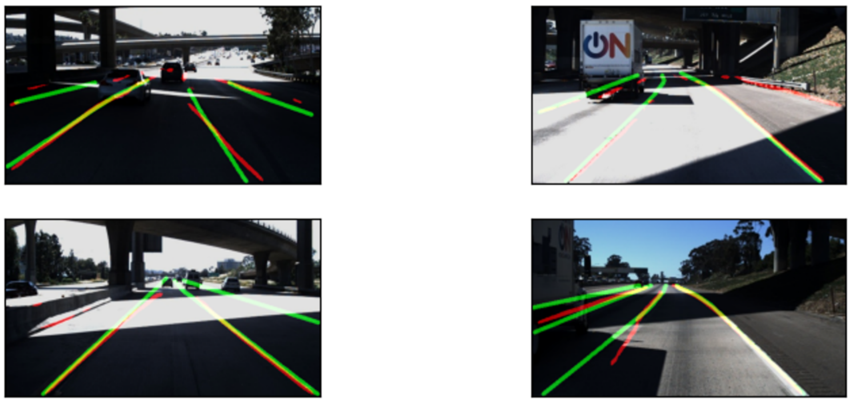

FIGURE 1.1: Lane detection using LaneNet [43] under driving scenarios with shadows, light reflections and lane occlusions from the TuSimple dataset. Green lanes represent the ground truth and red, the one inferred by the CNN.

figure 1.1. The reason might be that with the information contained in the actual frame, is very difficult or even impossible to completely inference the position of all lanes. Due to the fact that modern systems have to work under really variable driving scenarios, working under all possible conditions is crucial to develop robust algorithms.

1.1 Motivation

The general motivation of this thesis is to contribute to the research of a reliable solution towards a robust lane detection algorithm, combining the already available architectures to improve, or at least equalize, the state-of-the-art result in terms of:

1. Performance (specially focus in challenging situations like shadows, light reflections, lane occlusion, etc.).
2. Neuronal network size, which influences the needed computational resources to run the algorithm.
3. Running time, which is a key factor in embedded systems aiming real time performance.

1.2 Problem statement

This work focuses on the lane binary segmentation problem. “An image segmentation is the partition of an image into a set of non overlapping regions whose union is the entire image. The purpose of segmentation is to decompose the image into parts that are meaningful with respect to a particular application” [24]. So ultimately the output segmentation map indicates which pixels belong to a lane and which not. At this point it should be mentioned that during this thesis we work at pixel level, and that this problem definition should not be mixed with the instance segmentation problem, which takes care of disentangling the pixels identified as lanes and clustering them into the different lanes themselves.

The segmentation of an image I for a uniformity predicate P is a partition of I into disjoint non-empty subsets I_1, I_2, I_3, etc., I_n and can be defined mathematically as follows:

1. $\bigcup_{i=1}^{n} I_i = I$
2. I_i is a connected region; i = 1, 2, 3, . . . , n
3. $I_i \bigcap I_j = \emptyset$ for all i and j: $i \neq j$
4. $P(I_i)$ = True for i = 1, 2, 3, . . . , n

As already explained, in this case there are only two subsets, one for the pixels belonging to the lane and one for the pixels which not.

1.3 Hypothesis

As already demonstrated in [59], the use of recurrent neuronal networks on classical CNN architectures like UNet and SegNet can achieve state-of-the-art performance. Our hypothesis and the starting point of this work can be expressed as follow:

"The combination of more complex/specialized encoder/decoder architectures, rather than UNet or SegNet, with recurrent neuronal networks achieves or improves state-of-the-art performance on the problem statement formulated in section 1.2"

One of the architectures which we would like to try out is LaneNet [43] due to its classical encoder/decoder architecture and to its state-of-art performance (tested on TuSimple dataset).

1.4 Objectives and methodology

Our overall objective is to justify and validate our hypothesis from section 1.3. In order to reach such a goal, the following specific tasks have been defined:

1. Study current approaches on lane detection, specially the ones focusing on capturing time information.
2. Design and select different deep learning architectures.
3. Evaluate and compare their performance using the proposed metrics.

1.5 Document structure

The remainder of this thesis is organized as follows. In chapter 2 we describe the related work, not just focusing on deep learning, but also analyzing traditional computer vision methods. In chapter 3 the proposed, investigated and implemented architecture for semantic lane segmentation is presented. Experimental results are presented in chapter 4. Finally, chapter 5 discusses the obtained results and chapter 6 concludes this thesis and proposes further work.

Chapter 2

Literature review and related work

In the past decades quite a few number of researches have been done in the field of lane detection. These methods can be split mainly into two categories: classical computer vision methods and deep learning methods. Details of each architecture can be found below.

2.1 Classical computer vision methods

Most traditional computer vision methods follow normally a two step architecture consisting of: pre-processing/feature extraction and fitting/tracking modeling [55]. The most common methods during pre-processing include region of interest (ROI) selection, bird's eye view, also known as inverse perspective mapping (IPM), or color domain conversion. Once the images have been pre-processed, lane feature extraction, which is the fundamental step of the process, can be done. Finally, after lanes have been successfully detected, a model can be fit into the them and their position can be tracked with algorithms such as Kalman or particle filter to refine the inference and increase performance. Therefore, lane detection based on classical computer vision methods can be mainly organized by the pre-processing/feature extraction method and by the fitting/tracking model used.

1. **Pre-processing/feature extractions**: this kind of methods rely on feature detection based on color, texture or gradients. The idea of lane detection based on color is that, as proposed in [56], white and yellow can be more easily detected in other color domains because the contrast is increased. In [38] an extended edge linking algorithm is used to produce more complete edge-links, and features like lane-mark edge orientation and lane-mark width in YUV format are used to select candidate lane-mark edge-link pairs. In [13] images were converted to HSV to increase the contrast and an unsupervised and adaptive binary classifier based on the brightness values was proposed. [51] uses adaptive threshold to detect lanes based on pixel intensity and edge information. In general, color transformation is not robust and has limited ability to deal with shadows and illumination variation as stated in [55].

Another option is to detect lane features in the frequency domain [33]. Lane-finding in another domain (LANA) algorithm was proposed in [34] which captures lane strength and orientation in the frequency domain and final lane detection is done using template matching. Results showed that LANA was robust under varying conditions.

With the goal of removing camera distortions, bird's eye view perspective transformation can be used. This is proposed in [17] where after IPM transformation, and adaptive lane detection and classification method based on spatial lane features and Hough transform algorithm was used. [6] also detect lanes in bird's eye view with the help of Gaussian filter. Other studies focus on other types of filters like [58] in Steerable filter or [48] in Gabor filter. In general, bird's eye view has the inconvenience that is really sensitive to the calibratable parameters, which could even change during acceleration or deceleration of the vehicle. Another possibility for curve-fitting and distance estimation is using stereo vision [12][18], avoiding the calibration problems of bird's eye view, but requiring two cameras.

[14] proposed a method based on template matching to find possible lanes and color clustering to extract them, introducing deep learning architectures (multilayer perceptron network) for illumination-invariant recognition.

In general, lane detection using feature based methods do not need so much computational resources as others, and work well under certain conditions, but due to too many assumptions and constrains, they show a lack of robustness dealing with shadows and under poor visibility conditions [55].

2. **Fitting/tracking model**: once the lanes have been detected, different models have been used for curve fitting and improving performance, such as linear, parabolic or spline. Among these, spline lines is the one which offers more flexibility [55]. Hough Transformation (HT) is very common in the literature [7] for lane detection and fitting (just straight lines). In [54] Catmull-Rom spline was studied. To improve performance [16][57] proposed a B-snake curve fitting, which can describe any arbitrary shape by modifying the control points. [37] introduced parallel-snake model and [30] used a linear-parabolic model. [6] roughly detected the lanes with Hough transform and then improved it with Random sample consensus (RANSAC), which is the most popular algorithm to estimate the model parameters, and B-spline model. For the analysis, it also introduced the well known Caltech lane dataset.

   To even improve more the results, tracking techniques can be used. Kalman filter is widely studied in the literature [9][52][41]. Another very common filter is the particle filter [39]. [40] combined both filters in a Kalman particle filter achieving more robust results in simultaneous tracking of multiple lanes.

   In general, the use of model fitting can considerably improve the performance and robustness due to the fact that noise and outlier pixels can be ignored by the model.

On the other hand, these methods have more computational cost and are not so easy to implement [55].

2.2 Deep learning

Research in computer vision in general, and in lane detection in particular, has reached a new stage with the development and boom of deep learning. In the last decade many different architectures have been proposed, outperforming, in general, classical approaches. Other studies [59] classified them into three categories.

1. **Convolutional Neuronal Networks (CNNs) [35] with encoder/decoder architecture**: this type of network is very typical for semantic segmentation tasks. Two architectures commonly used are UNet [46] and SegNet [8]. These networks are proposed in [59] for lane detection and are also studied in this work. Based on SegNet, [43] proposed a network called LaneNet, specialized in real time lane detection, and used across this work. To avoid the need of huge expensive labeled data [11] exploited additional sensor types to generate large quantities of annotated images in a weakly-supervised way, which were then used to train a deep semantic segmentation network (UNet).

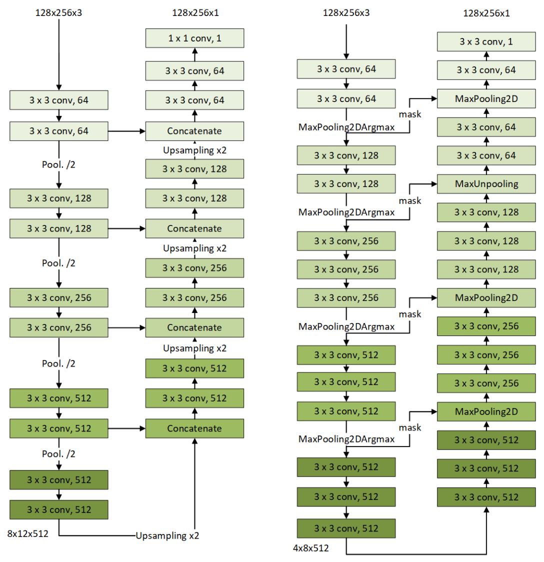

FIGURE 2.1: Encoder/decoder architecture. Left UNet and right SegNet. For simplicity reason batch normalization and activation layers after convolutional layers are not shown.

In [44] a spatial CNN (SCNN) was proposed, where traditional layer-by-layer CNNs were generalized to slice-by-slice CNNs, enabling message passing between pixels across rows and columns in a layer.

2. **Generative Adversarial Networks (GANs) [23]**: have also been used for lane detection. [21] proposed an embedding-loss GAN (EL-GAN). It follows a typical GAN

architecture, where the lane are predicted by a generator and judged by a discriminator.

3. **Convolutional Neuronal Networks (CNNs) [35] and Recurrent Neuronal Networks (RNNs) [47]**: this idea was first introduced in [36]. In this approach the RNN is not inferring along different frames, but along slices (different region of interest) of one frame. Then, a CNN extracts the features for each slice and, finally, a RNN is used to infer the lane from the feature maps of each slice. This method was reported to outperform results from only CNN based architectures. [59] proposed a new variant of a CNN and RNN network based on a encoder/decoder framework, which takes multiples frames as input and predicts the lane on the last one at pixel level. The CNN encoder extracts the feature maps from the different frames, which serve as an input to a ConvLSTM. Finally the CNN decoder is responsible for the reconstruction and prediction itself. This idea is also used in this work. To have a more detailed description on its utilization, please refer to chapter 3.

In conclusion, although deep learning based methods need more computational resources and large amount of labeled data, they outperform conventional methods and show more robustness under different driving scenarios [55] (no classical computer vision method is ranked on the leaderboard of any lane detection challenge/benchmark -TuSimple, CULane or BDD100K-). With the breakthrough and explosion of deep learning and specialized hardware, it is also expected that new real time capable architectures, which need less labeled data to be trained, will be developed in the next years.

Chapter 3

Proposed method and implementation

3.1 Basic network architecture

As already explained in chapter 2, [59] combined convolutional neuronal networks (CNN) and recurrent neuronal networks (RNN), more concretely long short-term memory (LSTM) [26] in one single architecture. Lanes are static, visible, continuous objects on the street, with a huge overlapping between frames, meaning a highly related and temporal correlation of the signals. These types of characteristics make them ideal for the type of neural networks mentioned above:

1. **Convolutional Neuronal Networks (CNN)** [35]: unlike fully connected layers, where each neuron in one layer is connected to every neuron in the next layer, making them sensitive to over-fitting, CNNs work recognizing patterns in the image, assembling them together in an increasing complexity hierarchy. CNNs were inspired by biological processes where different groups of neurons react to different patterns [29]. This makes them very usable for computer vision tasks. CNNs architecture is at the end built upon recursive layers of convolution and pooling operations. Therefore an input image can be encoded or abstracted to a feature map containing the relevant information. [22].

2. **Recurrent Neuronal Networks (RNN)** [47]: lane's continuity and overlapping makes them very suitable for using time-series frameworks. RNN is a type of architecture known for its talent in continuous signal processing and sequential feature extraction. RNN shares the same weights across several time steps making it possible to generalize along different lengths (in this case different number of input frames).

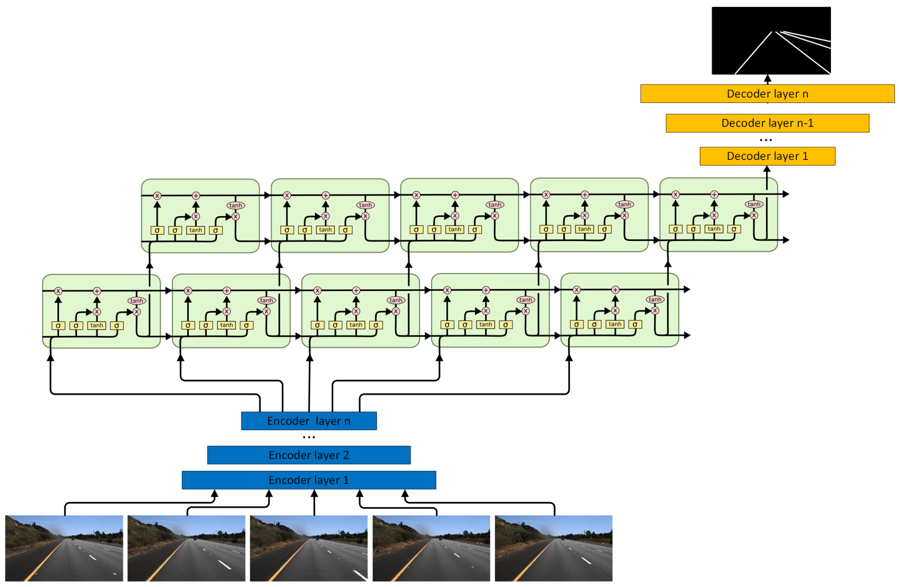

FIGURE 3.1: Network architecture for the special case of 5 frames and ConvLSTM2D with 2 layers. The basic idea of this architecture for lane detection was proposed in [59], however the encoder/decoder can be adapted and further changes can be done to improve not just performance but also other aspects, like running time or size.

The proposed architecture in [59] can be understood as an fully convolutional neuronal network with a recurrent intermediate step to process the time information. The encoder abstracts n-input images to n-feature maps which can be seen as time sequence information and be fed into the recurrent neuronal network. These feature maps contains the necessary information to detect the lanes and keep the time related information but have the advantage of its reduced size which means that they can be handled well by the long short-term memory layers. The output of the LSTM is used as input to the convolutional neuronal network decoder which outputs an array of the same size as the input image containing the probability of each pixel belonging to a lane or not. The complete architecture is shown in Figure 3.1. In this example five frames are used as input with a two layer ConvLSTM.

As already explained above, the inputs to the RNN blocks are the feature maps output by the CNN encoder. There are different types of RNN available in the literature like gated recurrent units (GRUs) [15], but LSTM has been proposed because they generally outperform other RNN architectures [20], partially thanks to its forget gate given them the ability to forget irrelevant unimportant information and remember essential features. The traditional LSTM is time- and computation- consuming, therefore, the proposed network uses convolutional LSTM (ConvLSTM) [49]. The ConvLSTM replaces the matrix multiplication in every gate of LSTM with a convolution, which is widely used in end-to-end training and feature extraction from time-series data. The equation describing the behavior of a ConvLSTM at time t can be formulated as:

$$C_t = f_t \circ c_{t-1} + i_t \circ tanh(W_{xc} * X_t + W_{hc} * H_{t-1} + b_c) \tag{3.1}$$

$$f_t = \sigma(W_{xf} * x_t + W_{hf} * W_{t-1} + W_{cf} \circ C_{t-1} + b_f) \tag{3.2}$$

$$o_t = \sigma(W_{xo} * x_t + W_{ho} * W_{t-1} + W_{co} \circ C_{t-1} + b_o) \tag{3.3}$$

$$i_t = \sigma(W_{xi} * x_t + W_{hi} * W_{t-1} + W_{ci} \circ C_{t-1} + b_i) \tag{3.4}$$

$$H_t = o_t \circ tanh(C_t) \tag{3.5}$$

where X_t denotes the input feature maps extracted by the encoder CNN at time t. C_t, H_t and C_{t-1} denote the memory and output activations at time t and t-1, respectively. C_t, i_t, f_t and o_t denote the cell, input, forget and output gates, respectively. $W_x i$ is the weight matrix of the input X_t to the input gate, b_i is the bias of the input gate. The meaning of other W and b can be inferred from the above rule. $\sigma()$ represents the sigmoid operation and $tanh()$ represents the hyperbolic tangent. $*$ denotes the convolution operation and $\circ$ the Hadamard product.

3.2 Encoder/Decoder selection

One of the advantages of this architecture is its flexibility due to the fact that it can be adapted to be used with different types of encoder/decoder networks. Although the original paper [59] focuses on UNet [46] and SegNet [8], we have decided to center ourselves on another encoder/decoder architecture, LaneNet [43]. We hope that this approach can outperform the original paper under some circumstances due to:

1. LaneNet is and encoder/decoder architecture which is already specialized in lane detection. Although LaneNet shows state-of-the-art performance, it has a lack of robustness when dealing with challenging situations (Figure 1.1).

2. LaneNet has fewer number of parameters and is a smaller network, making it more suitable for real time capable applications. Figure 3.2 shows the number of trainable parameters depending on the selected encoder/decoder architecture. LaneNet-LSTM has approximately 80 times fewer parameters than UNet-LSTM and 102 times than SegNet-LSTM. Although these numbers do not tell anything about the performance or the real time capability of the network, they do provide information about how much memory is needed to store them.

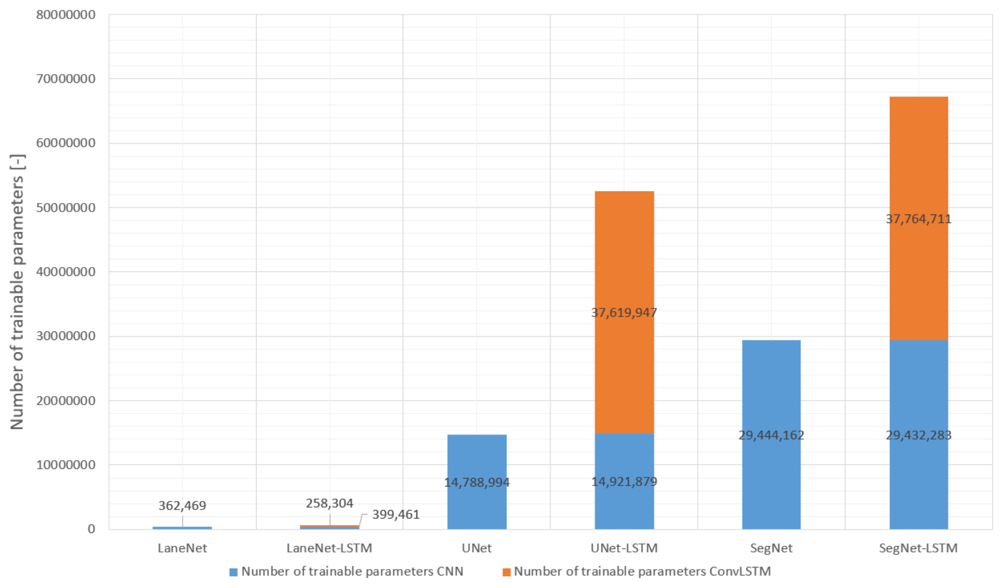

FIGURE 3.2: Number of trainable parameters for the different architectures (encoder and decoder variations) used during this work. It is just an indicative due to the fact that small changes in the LaneNet-LSTM have been done to try to improve its performance. As basic for this figure, a 2-layer and 32-filter ConvLSTM is used. Another take away from this figure is that ConvLSTM represents 39.3% of the trainable parameters in the LaneNet-LSTM, 71.6% in the UNet-LSTM and 56.2% in the case of SegNet-LSTM.

3.3 Variations

Until now we have explained the basic idea of the architecture to be used in this work and why, we think, it has the potential to outperform the state-of-the-art algorithms. However, during training we observed high fluctuations on the validation loss (Figure 3.3). In general, this might be a typical behavior when training deep learning models and might be explained because the training loss is actually the target of the training process and it is therefore affected directly, while the validation loss is only affected indirectly, so it is more volatile. However, this could also point to some other underneath problems:

1. **Network size**: one of the conclusion of Figure 3.2 is not just the number of parameter themselves, but also the proportion between the encoder/decoder part and the ConvLSTM. In the case of the LaneNet-LSTM, 39.3% of the trainable parameters belong to the ConvLSTM. This percentage is increased to 71.6% in the case of UNet-LSTM and to 56.2% in the case of SegNet-LSTM. In order to distribute homogeneously the number of parameters over the network, we introduced a 2D convolutional layer at the LSTM output, making its output smaller, reducing the number of parameters (filters) and keeping the structure of the decoder intact.

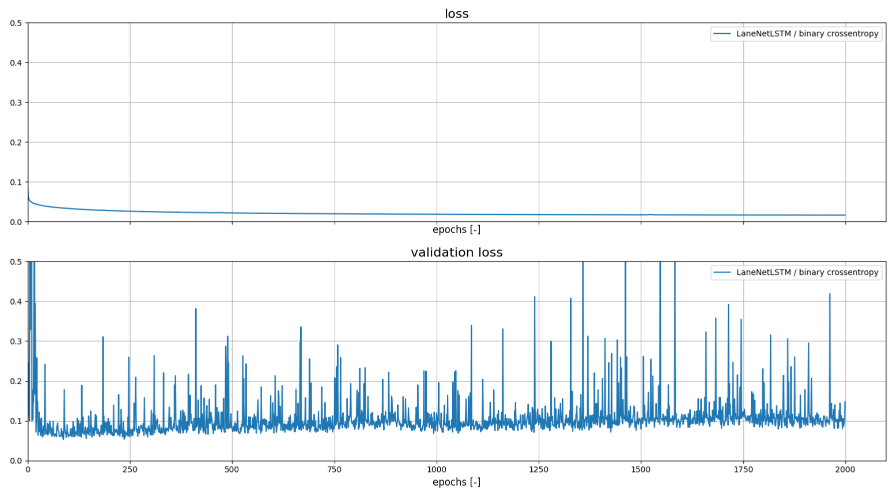

FIGURE 3.3: Loss and validation loss for LaneNet-LSTM. Fluctuations on the validation loss could indicate over-fitting problems. To try to mitigate this changes in the network size, regularization and batch size have been proposed.

2. **Regularization (dropout)**: other method very well known in the literature to mitigate over-fitting problems is regularization. In this case, we proposed dropout layer at the LSTM output to try to generalize better over the different time frames.

3. **Batch size**: during training the model is attempting to estimate the real distribution of the data, however it just gets the distribution of the training dataset (which should be similar but it must not be necessarily the same). In addition to this, when using small batch sizes, the distribution of the data is built upon a really small part of the training dataset, which makes this calculation even much more sensitive. This could lead to fluctuation in the losses which are not even related to the architecture itself. Therefore, the batch size is an important hyper-parameter. The problem of increasing the batch size is that the needed GPU memory increases linearly. In our case, where we deal with already very big inputs (n frame sets), very small batch sizes must be used in order to avoid memory allocation errors.

Chapter 4

Experimental results

In this chapter, the steps followed to achieve the methodology described in chapter 3 are presented. First, the dataset used and the pre-processing applied to it. Secondly, the implementation and equipment used for the training and the different hyper-parameters analyzed. And finally the metrics and key performance indicators used to evaluate the accuracy and robustness of the proposed method and its comparison with diverse lane detection state-of-the-art methods.

4.1 Dataset

For the training and key performance indicator the TuSimple dataset is used [2]. It contains 3626 video clips for the training set and 2782 for the testing set taken under good and medium weather situations, at different daytime and traffic conditions on US highways. Each video clip is a set of 20 frames with until 5 lanes, where just the last frame is labeled, and with a resolution of 1280×720.

Dataset	Train clips	Test clips	Frames pro clip*	Resolution	Lanes	Scenarios
TuSimple	3626	2782	20*	1280x720	$\leq$5	US Highways

TABLE 4.1: Characteristic of TuSimple dataset [2]. * just the frame 20th is labeled.

The annotation of each lane is done using polylines, defined by the intersection points (represented as green circles in Figure 4.1) between each evenly horizontal distributed red line and each lane. This means that at most five polylines might be defined for each frame. On the other hand, the proposed architecture is conceived to work at pixel level, classifying each as lane or not lane. Therefore, using these polylines, ground truth frames have been generated where each pixel belongs to the class 0 (background) or 1 (lane). Each lane has a width of 5 pixel -for 512x288 resolution- and 2 pixel -for 256x144 resolution-. Although the architecture proposed in chapter 3 is quite flexible, the encoders/decoders have been designed for a specific resolution and this is the reason why we have generated two different ground truths with two different resolutions. But, in any case, the lane width is approximately 1% of the total image width.

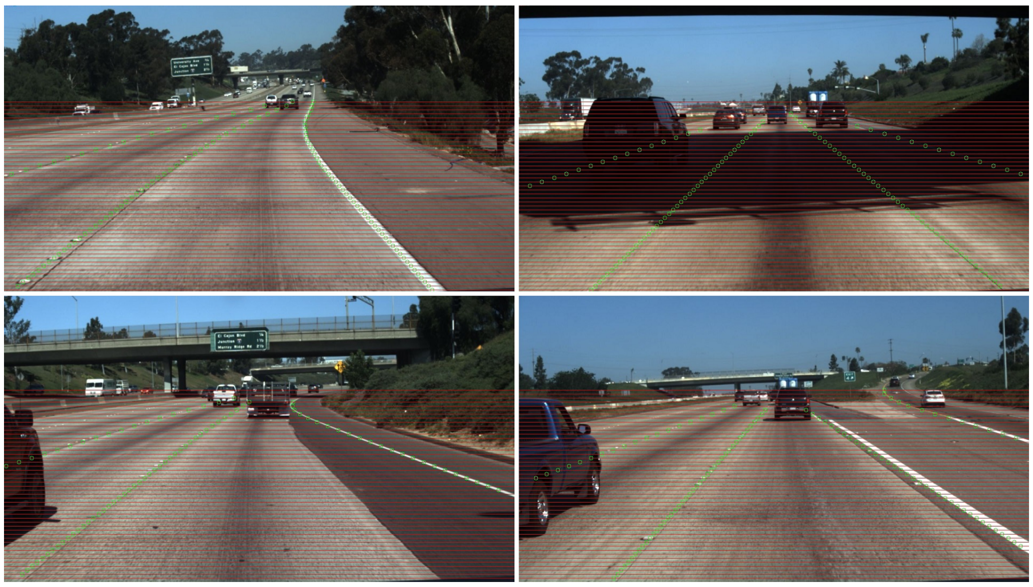

FIGURE 4.1: Example of TuSimple dataset [2]. Even distributed horizontal lines are plotted in red and the intersection with each polyline with a green point.

In order to avoid over-fitting problems, data augmentation has been used to train all models. Many studios have shown that transformations in the color scale, bright and gamma augmentation, help generalization [50]. Many researches have also proved that adding small amount of noise (jitter) to the training data usually has the same effect on regularization and avoid over-fitting problems [45]. Therefore these kinds of data augmentation have been used throughout this work. In addition to this, also random horizontal flip and translation has been implemented.

4.2 Implementation

The experiments conducted during this work were executed on a computer equipped with an Intel(R) Xeon(R) E5-1650@ 3.50GHz, 126 GB RAM and four Nvidia GeForce GTX 1080 with 11 GB RAM and 11,47 peak TFlops (SP).

This work has been implemented in Python using Tensorflow 2.0 [3] and Keras 2.3.1 [4]. Tensorflow is a software library created by Google which used computational graphs to describe machine learning algorithms. A computational graph is a type of graph where the nodes describe operations and the edges describe data input to or output by those operations. The main advantage of Tensorflow, and in general similar frameworks, is that we just have to development the forward pass, and the backward pass (needed for gradient descent optimization) is calculated automatically. Instead of writing the code directly on Tensorflow we have used the high-level neural networks API Keras.

4.3 Metrics

Lane detection is an imbalanced binary classification problem, where the amount of ones, which represent lanes, are much less than the amount of zeros, which represent the background. In general, ones are just 4% of the dataset, this means that classifying all pixels as no-lane gives an accuracy of 96%. Therefore, accuracy is just a reference index, and it should not be used as key performance indicator. Therefore precision, recall and false positive rate are metrics which can indicate with higher precision the performance of the system:

$$precision = \frac{TP}{TP + FP} \tag{4.1}$$

$$recall\ or\ true\ positive\ rate = \frac{TP}{TP + FN} \tag{4.2}$$

$$false\ positive\ rate = \frac{FP}{FP + TN} \tag{4.3}$$

$$F1 = 2\frac{precision \cdot recall}{precision + recall} \tag{4.4}$$

where TP stands for true positive, TN true negatives, FP for false positive and FN for false negative. Evaluating these metrics under different thresholds, ROC (receiver operating characteristic curve) and PRC (precision recall curve) can be generated. Using these curves offers the advantage of analyzing the classifier using the whole spectrum rather than selecting one arbitrary threshold. To calculate the true positive rate and false positive rate used for ROC, TP, TN, FP and FN are used. This means that ROC does not show any bias toward models that perform well on the minority (1) class at the expense of the majority (0) class [25]. Although ROC can be used for imbalanced dataset, the results might be at some point misleading due to the fact that a small number of correct or incorrect predictions could lead to big changes on ROC and hence on the area under it. On the other hand, both precision and recall are not influenced by TN, which represent the majority (0) class in this case, meaning that PRC focuses on the minority class and therefore might be prefered when dealing with imbalanced datasets where ROC curves may provide an excessively optimistic view of the classifier performance [10].

The above mentioned metrics are suitable for segmentation tasks. For the performance calculation at lane level, which exceeds the scope of this work, a common metric used in the literature is the accuracy calculated as the average correct number of lane's points per image:

$$accuracy = \sum_{im} \frac{C_{im}}{S_{im}} \tag{4.5}$$

with C_{im} the number of correct points and the S_{im} the number of ground truth points. A predicted point is said to be correct when the horizontal difference to the ground truth is less than a threshold [2].

4.4 Training

Once the architecture is implemented, the neuronal network can be trained to infer the ground truth by the use of back propagation. During training the following aspects have been considered.

1. **Loss function**: normally in this type of binary classification problem, binary cross entropy is used, because is easily derived, which is necessary to apply gradient descent during the training phase. Sometimes, when working with an imbalanced dataset, as it is the case, also weighted binary cross entropy is very common in the literature to avoid the fact that classifying all pixels as no-lane leads to a really high accuracy (in this case of approximately 96%). During this work both loss functions have been analyzed.

$$binary\ cross\ entropy = \sum_{i=1}^{n} y_i * log(\hat{y}_i) + (1 - y_i) * log(1 - \hat{y}_i) \tag{4.6}$$

$$weigthed\ binary\ cross\ entropy = \sum_{i=1}^{n} C_0 * y_i * log(\hat{y}_i) + C_1 * (1 - y_i) * log(1 - \hat{y}_i) \tag{4.7}$$

$$C_0 = \frac{num_{samples==0}}{num_{samples}} \quad C_1 = \frac{num_{samples==1}}{num_{samples}} \tag{4.8}$$

where n is the number of pixels, y_i is the ground truth at pixel i, $\hat{y}_i$ is the predicted value at pixel i and C_0 and C_1 are constants which depend only on the imbalanced dataset. One disadvantage of these kind of loss functions is that they make no distinction between 1-pixel error or n-pixels error. As shown in figure Figure 4.2 both predictions (green and red) lead to the same binary cross entropy error, although one of them is much better, closer to the actual ground truth.

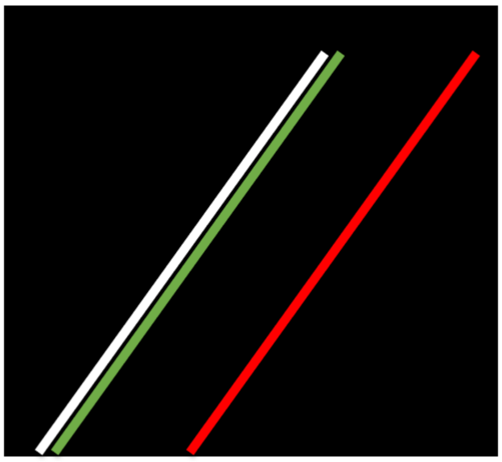

FIGURE 4.2: Ground truth lane is represented with the color white. Lanes with color green and red are two possible predictions with the same binary cross entropy error, although the green one has an smaller "error".

Our hypothesis is that, at some point, this could difficult the convergence to the global/local minima. In other to tackle such a problem, we propose a small modification to the function loss, smoothing the ground truth by applying a Gaussian filter to it. Mathematically, this can be expressed as follows:

$$soft\,binary\,cross\,entropy = \sum_{i=1}^{n} G(y_i) * log(\hat{y}_i) + (1 - G(y_i)) * log(1 - \hat{y}_i) \tag{4.9}$$

where $G(y_i)$ represents the Gaussian filter of the ground truth at pixel i.

In table Table 4.2 the influence of the different loss functions on LaneNet is presented. As shown, none of the proposed functions have an statistical impact on the network performance. Even weighted binary cross entropy, which is really common in the literature, does not lead to further improvements.

Network	**AUROC**	**AUPRC**	**Accuracy**	**F1**
LaneNet/bin. cross entropy	99.276	87.616	98.291	78.832
LaneNet/weighted bin. cross entropy	99.224	87.045	98.259	78.422
LaneNet/soft bin. cross entropy/fil. size=3	99.280	87.625	98.288	78.819
LaneNet/soft bin. cross entropy/fil. size=5	**99.326**	**87.986**	**98.304**	**78.983**
LaneNet/soft bin. cross entropy/fil. size=7	99.265	87.537	98.286	78.786
LaneNet/soft bin. cross entropy/fil. size=11	98.286	87.510	98.279	78.658
LaneNet/soft bin. cross entropy/fil. size=11; 7; 5; 3; 0	98.285	87.751	98.306	79.102

TABLE 4.2: Influence of the different loss functions on the area under ROC (receiver operating characteristic curve), under PRC (precision recall curve), maximum accuracy and its corresponding F1.

In figure Figure 4.3 the training and validation loss for the different loss functions on LaneNet over the epoch is presented. The proposed soft binary cross entropy does not show any effect on the convergence speed either.

According to this analysis binary cross entropy is a really stable and robust loss function and therefore it has been used across the comparison between the different variants of LaneNet-LSTM.

2. **Optimizer**: we have used ADAM with a learning rate of 3e-4. ADAM is much less sensitive to hyper-parameters, including a bad learning rate, than other optimizers. Although, in general, a well calibrated stochastic gradient descent (SGD) would probably slightly outperform ADAM when working with convolutional neuronal networks, its optimal learning rate region is much more narrow and problem-specific. In addition, the used of ADAM is much more common when working with RNNs [5].

3. **Batch size**: batch size is an important hyper-parameter which could help to mitigate the high fluctuation on the validation loss. On the other hand, as already explained in chapter 3, higher batch sizes mean higher memory needs, and due to

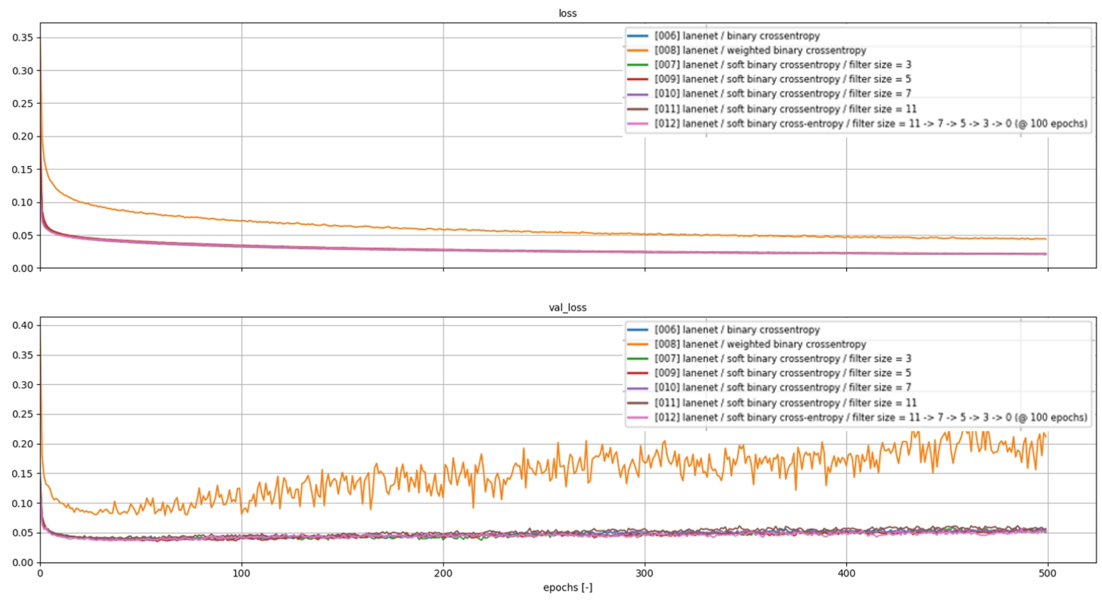

FIGURE 4.3: Training and validation loss for the different proposed loss functions. Soft binary cross entropy does not show any effect on convergence speed.

hardware limitations, no increase of it was technically possible. So we have used a batch size between 6 and 15, depending on the number of input frames.

4. **Epochs**: each of the proposed networks have been trained until 500 epochs, although, in most cases, from epoch 200 signs of over-fitting start to appear.

4.5 Performance

In table Table 4.3 the final result of the proposed architecture and its variants is presented. We have investigated the influence on performance of the following factors:

1. **Filters**: in order to use the decoder without modifications, the output of the recurrent neuronal network have to match. This could lead, as already explained in chapter 3, to relatively big ConvLSTM. To avoid this, a standard convolution neuronal network, which map the RNN output to decoder input can be added, leading to a decrease of the network trainable parameters without affecting the performance (from 2,722,789 trainable parameters with 128 filter to 657,765 with 32 filters).

2. **Frames and stride**: number of input frames and their stride. The proposed architecture gives many possibilities regarding the inputs, not just about how many input frames, but also their stride. In this way, not just the covered space/time, but also the overlapping factor can be considered as hyper-parameters of the network. Table 4.3 shows how increasing the number of frames, leads to approximately 0.1% increase in performance (at the cost of needing more computational resources for the encoder).

3. **Dropout**: dropout layer at the RNN output. As already explained in chapter 3, big fluctuations on the validation loss could indicate over-fitting problems. In order to try to mitigate them, dropout layer at the RNN output has been proposed to try to stimulate generalization on the feature map. Table 4.3 shows how a slightly increase of the dropout rate is beneficial for the network performance, reducing thus over-fitting. At some point, an excessive dropout has the opposite effect, leading to performance drop because of reducing the network's degree of freedom too much (under-fitting).

Filters	Dropout	Frames	Stride	AUROC	AUPRC	Accuracy	F1
128	0.0	5	1	99.186	86.229	98.206	77.727
128	0.0	5	2	99.270	87.471	98.280	78.687
128	0.0	5	3	99.261	87.287	98.268	78.602
128	0.0	2	1	99.149	86.126	98.227	78.090
128	0.0	2	2	99.156	86.407	98.234	78.231
128	0.0	2	3	99.166	86.462	98.231	78.075
32	0.0	5	1	99.141	85.506	98.175	77.444
32	0.1	5	1	**99.326**	**87.947**	**98.319**	**79.144**
32	0.1	5	2	98.283	87.823	98.301	78.930
32	0.2	5	1	99.283	87.569	98.289	78.852
32	0.3	5	1	99.240	87.160	98.276	78.627

TABLE 4.3: Area under ROC (receiver operating characteristic curve), area under PRC (precision recall curve), maximum accuracy and its corresponding F1 for the different combinations of LaneNet-LSTM. The results correspond to the better results obtain for each architecture.

4.6 When is the network really over-fitting?

To be able to select the optimal model, one key question to be answered is: when is the model really over-fitting? Although this might seen as a quite easy question to answer, it might still have some open points. At the beginning, we can just try to optimize the validation loss, and pick out the best model just depending on it. Although this might be seen as a reasonable approach, it might be sometimes not the best decision. During the training process of some of the proposed architectures we have seen that although the validation loss increases, other metrics like validation accuracy, validation precision or validation recall also increase (Figure 4.4). This rises an interesting question: which metric is more important? Should we focus on the validation loss or is the accuracy more important?

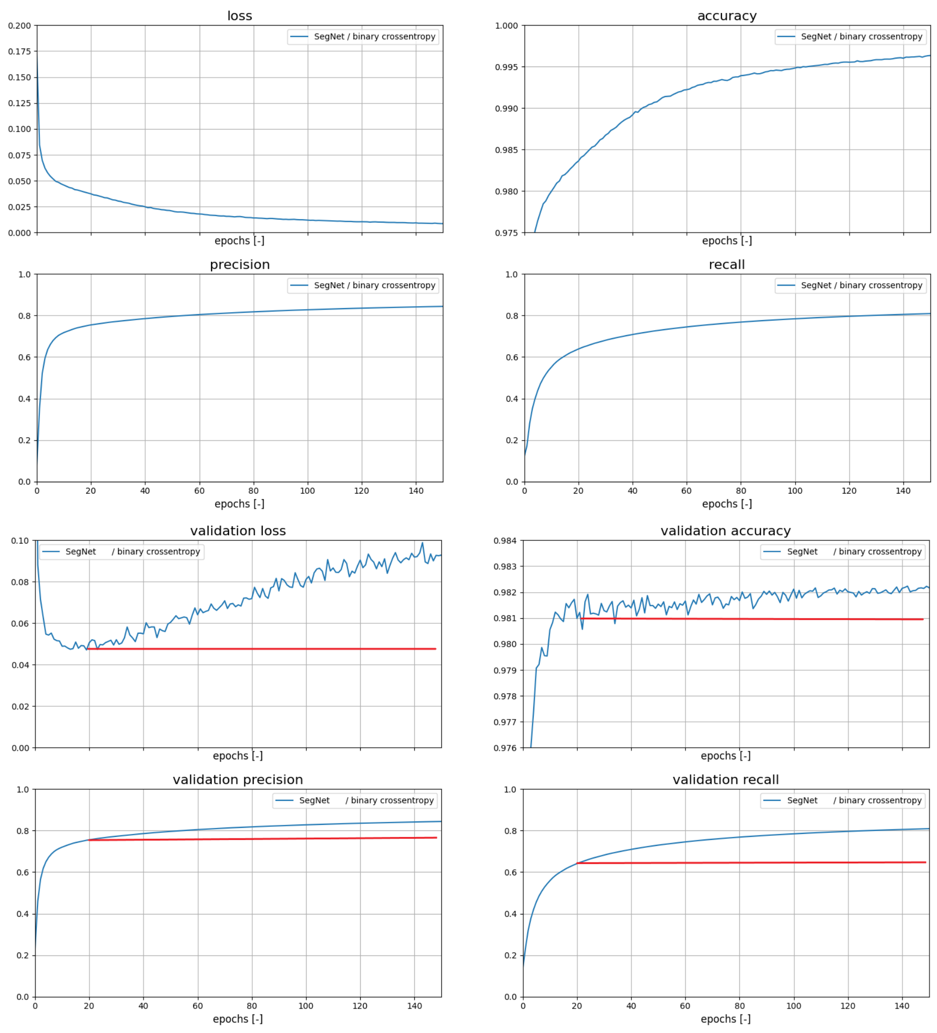

FIGURE 4.4: During the training of some of the networks we have seen the following phenomena: validation loss, validation accuracy, validation recall and validation precision increase simultaneously. Intuitively the relationship between validation loss and the other three metrics should be inverse.

Attending at their definition loss and accuracy does not measure exactly the same thing. Loss measures the difference between ground truth - which is 0 or 1 - and prediction - which is a floating number between 0 and 1. In the other hand, accuracy measures the difference between thresholded prediction - which is not a floating number anymore, but rather 0 or 1 - and the ground truth itself. According to this definition if the prediction changes, the loss will change, but the accuracy could stay constant until the predicted value goes over or under the defined threshold. This means that accuracy is more robust to changes in the raw prediction. However, intuitively there must be somehow a inverse correlation between both, due to the fact that better predictions should lead to lower loss and higher accuracy, and vice versa. And therefore the presented case is somewhat contra intuitive. The more expected behaviors would be:

1. **Loss decreases and accuracy increases**: this is the classic behavior at the beginning of the training, where the predicted value crosses the threshold leading to an

accuracy increase.

2. **Loss decreases or increases and accuracy stays the same**: this happens when the predicted value changes, affecting directly the loss, but this change is not enough to cross the threshold, meaning no change in the accuracy.

A possible explanation to this phenomena would be that during training the network learns new patterns, some of them are useful for the generalization and some of them are not. Learning a new pattern is the process when the predicted value crosses the accuracy threshold, leading to an actual change on the thresholded prediction. By definition, if the accuracy increases we are learning more useful patterns than patterns which lead to over-fitting problems. On the other hand, learning more useful patterns does not mean necessarily that the validation loss decreases. In fact, when we secure incorrect patterns, in the sense that the predicted value goes incorrectly whether towards 0 or 1, the validation loss blows up. This is because the binary cross entropy has a lower limit, 0 - $\lim_{x\to 1} log(x) = 0$ -, but no upper limit - $\lim_{x\to 0} log(x) = -\infty$ -.

So getting back to the original question, when is the network really over-fitting? From a definition point of view, the network is over-fitting because the validation loss increases, but it is at the same time learning more useful patterns than incorrect ones.

4.7 Comparison with state-of-the-art

4.7.1 Quantitative performance: pixel level

In this section we compare the proposed architecture with other state-of-the-art methods. As we are mainly focus on the segmentation task in this work, all the metrics we use are at pixel level. In Table 4.4, the results for the different architectures in terms of area under receiver operating characteristic curve, area under precision recall curve, maximum accuracy and its associate F1 are presented. These metrics are the average of five trainings to be able to make better statistical conclusions.

Network	**AUROC**	**AUPRC**	**Accuracy**	**F1**
LaneNet	99.276	87.616	98.291	78.832
LaneNet-LSTM*	99.274	**87.685**	98.300	78.965
UNet	98.097	85.954	98.416	78.528
UNet-LSTM	**99.369**	87.654	**98.450**	**79.095**
SegNet	98.353	83.022	98.186	74.806
SegNet-LSTM	99.122	84.476	98.215	75.287

TABLE 4.4: Area under ROC (receiver operating characteristic curve), under PRC (precision recall curve), maximum accuracy and its corresponding F1 for the compared state-of-the-art networks. * refers to LaneNet-LSTM with 32 filters ConvLSTM2D and dropout of 0.1.

As a consequent of running the training multiple times, the result can be also presented in a box diagram (Figure 4.5). Finally, to test whether the proposed method offers a statistical improvement with respect to the state-of-the art, we have used the Student's t-test (Appendix A). At 5% significance level, we do not reject the hypothesis that the mean of the populations is the same, or there is no sufficient evidence in the data to conclude that the populations are different. The final conclusion is thus, that the three architectures LaneNet, LaneNet-LSTM and UNet-LSTM have statistically the same performance.

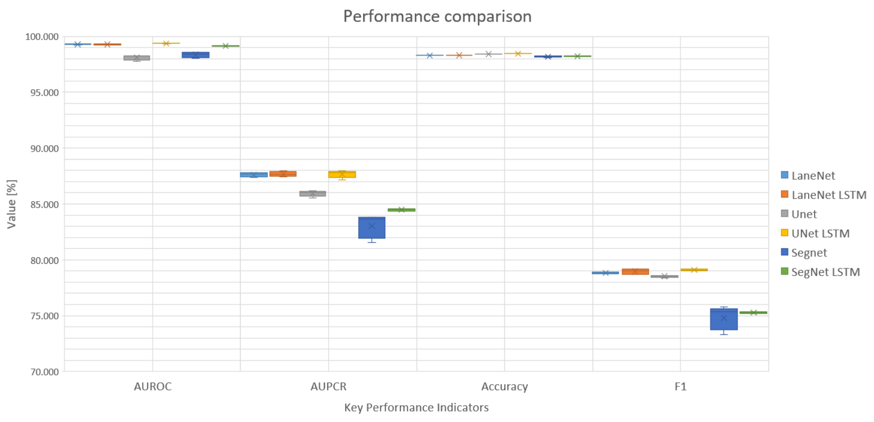

FIGURE 4.5: Box plots for area under ROC (receiver operating characteristic curve), under PRC (precision recall curve), maximum accuracy and its corresponding F1 for the compared state-of-the-art networks.

One important point of this analysis is that the TuSimple dataset has been taken mostly under favorable conditions, no shadows, no lane occlusions, etc. On the other hand, the discussed methods should outperform single-frame detection algorithms specially under challenging situations, therefore the proposed metrics have been recalculated just for these situations (to do so a manual classification of the validation set has been done, Appendix B). The results of this analysis are presented in Table 4.5 and show that multiple-frame detection algorithms clearly outperform single-frame ones under challenging situations.

Network	AUROC	AUPRC	Accuracy	F1
LaneNet	96.487	67.234	96.724	57.437
LaneNet-LSTM*	97.282	70.445	96.877	60.943
UNet-LSTM	**97.627**	**72.519**	**97.278**	**64.106**

TABLE 4.5: Area under ROC (receiver operating characteristic curve), under PRC (precision recall curve), maximum accuracy and its corresponding F1 for the compared state-of-the-art networks for only the challenging situations contained in the TuSimple validation set. * refers to LaneNet-LSTM with 32 filters ConvLSTM2D and dropout of 0.1.

Although in subsection 4.7.3 a more general visual examination is presented, we would like to briefly introduce here the results in adverse scenarios. Semantic segmentation architectures should work robustly at a coarse level, identifying the total number of lanes correctly, and at a fine level, detecting solid and robust lanes with a high overlapping with the ground truth. With theses two requirements in mind, the experimental results show that ,in general, multiple-frame architectures outperform single-frame ones, and, in particular, that the proposed algorithm outperform state-of-the-art single-frame networks. Figure 4.6 shows three examples of the predictions of the above mentioned architectures (LaneNet, LaneNet-LSTM, UNet-LSTM) under challenging situations.

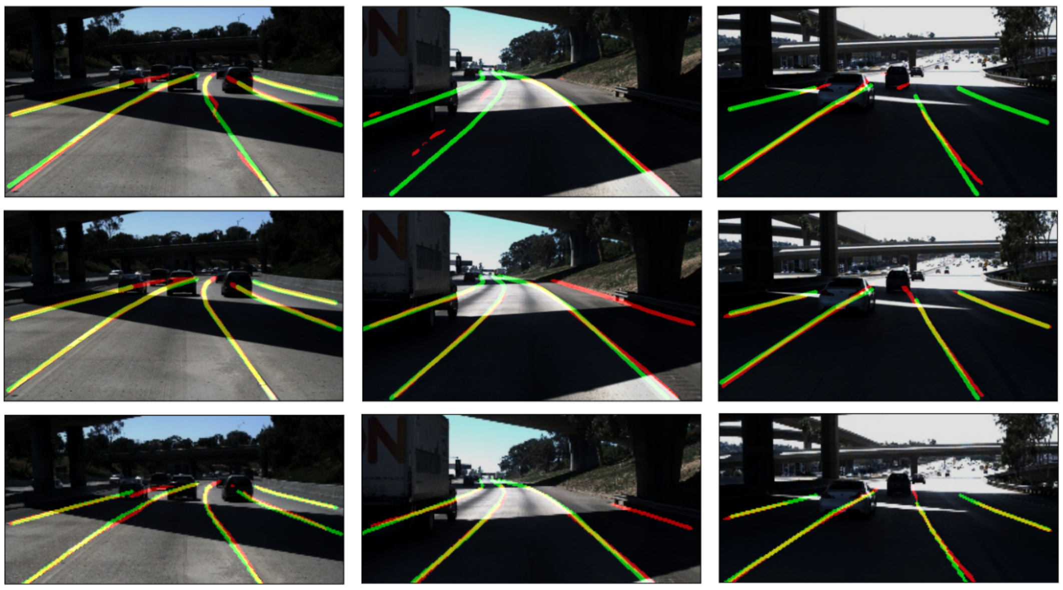

FIGURE 4.6: Three examples of images in challenging situations. Top row: LaneNet. Middle row: LaneNet-LSTM. Botton row: UNet-LSTM. Green lanes correspond to ground truth and red lanes to predictions (yellow is just an overlap of both of them).

Multiple-frame architectures show a more robust behavior, inferring more solid and robust lanes specially when the contrast is changing rapidly (e.g. entrance of a tunnel). Under this challenging situations, they also show higher capabilities on detecting the lines of the neighboring lanes. The proposed LaneNet-LSTM architecture is also even able to correctly inference complex side lines, which are not in the ground truth, decreasing thus the quantitative performance.

4.7.2 Quantitative performance: lane level

Although the main focus of this work is lane segmentation, we have also investigated one possibility to analyze the results at lane level. The initial concept proposed can be described in two mayor steps:

1. **Calculate binary segmentation map with one of the already presented architectures**: this is the main focus of this work and the used metrics and obtained results have been already presented in the subsection 4.7.1.
2. **Use the Hough transform [19] to extract the lane features**: can be used to only describe straight lanes. It is a paremetric method where the distance resolution of the accumulator, the angle resolution of the accumulator and accumulator threshold

can be calibrated. The main problem with this method is that the Hough transform is not working really stable on the segmentation maps, being very sensitive to parameters change. In order to detect every segmented lane, the accumulator threshold has to be so far reduced, that some lanes are found multiple times and even non-existing one are detected like in Figure 4.8.

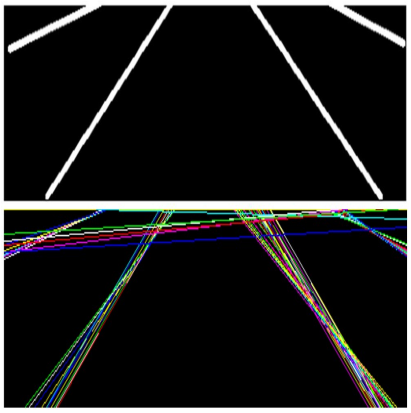
FIGURE 4.7: Example result of the Hough transform applied to the segmentation map.

Because of the fact that Hough transform does not work as expected, we have research also another method based on region growing. The segmentated pixel are clustered into different lines using region growing and finally a line is fitted into the different cluster. Although this method works well when the segmentation is solid and robust, it is not able of performing in real time - it takes 4 second for an image of 512x256 pixels -.

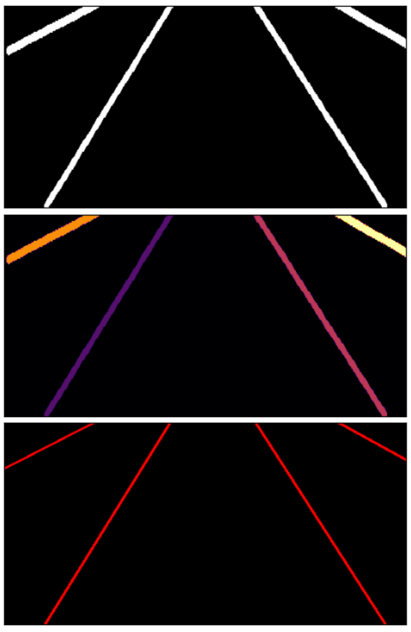
FIGURE 4.8: Example result for region growing. Top: raw image. Middle: output of the clustering. Bottom: result after line fitting.

Although the main focus of this work is not the detection of the lanes themselves, but just the segmentation part, small investigation using traditional computer vision algorithms has been done. As already discussed, even the breakthrough from segmentation to lanes is not a straightforward step, and it requires complex algorithms in order to work robustly in all possible segmentation maps. Analyzing this step could be a new research line for future works.

4.7.3 Qualitative performance

Until now we have mostly analyzed the performance just in terms of metrics, but it is always a good practice to understand visually where the networks are having more problems to figure out possible weak points and improvements. In order to do this, in Figure 4.9, Figure 4.10 and Figure 4.11 we have plotted the 20 worst predictions based on AUPRC for each architecture. In the images green lanes correspond to ground truth and red lanes to predictions (yellow is just an overlap of both of them). From this analysis we would like to highlight:

1. As stated in chapter 1 one of the main goals of the proposed method is to improve performance under driving scenarios with shadows, light reflections and lane occlusions. Among the 20 worst predictions, while for LaneNet there are 7 frames under these circumstances, for LaneNet-LSTM and UNet-LSTM there are just 4 and 3 respectively. This is notable improvement of the performance under these challenging circumstances and might show that the Conv-LSTM is able of learning patterns from temporal information. This behavior matches the metrics and conclusions already presented in subsection 4.7.1.

2. Another point which has influenced the quantitative analysis is that there are some good predicted lanes which are not labeled in the validation set. Of course, the trained network cannot get better than the labels themselves, and this might be indicating that the labeling is not representing the ground truth with fidelity.

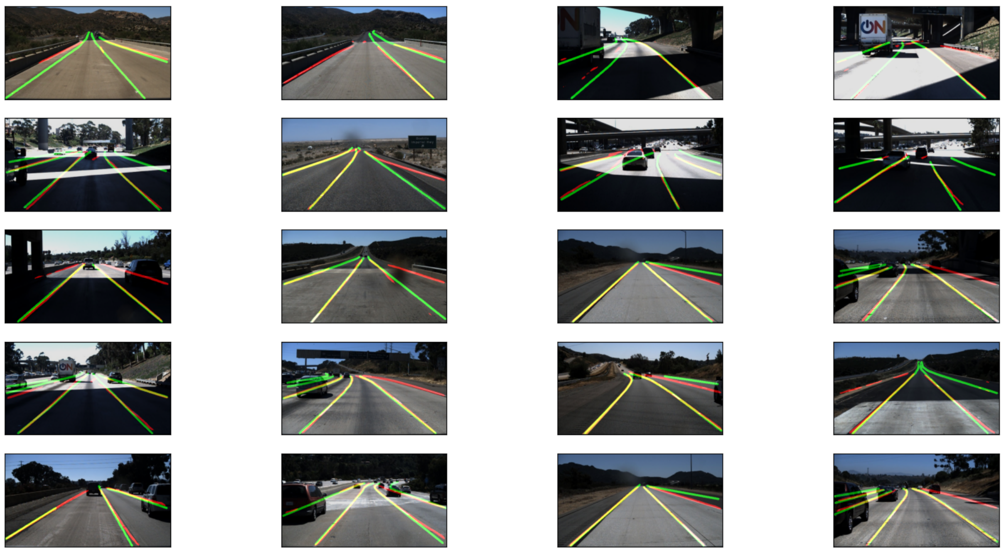

FIGURE 4.9: The 20 predicted scenarios with the worst AUPRC calculated by LaneNet.

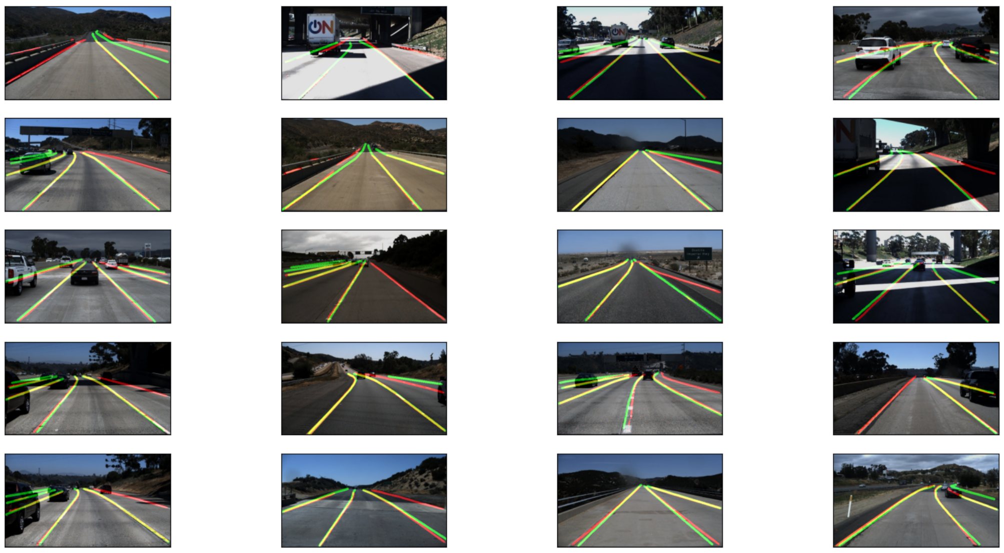

FIGURE 4.10: The 20 predicted scenarios with the worst AUPRC calculated by LaneNet-LSTM.

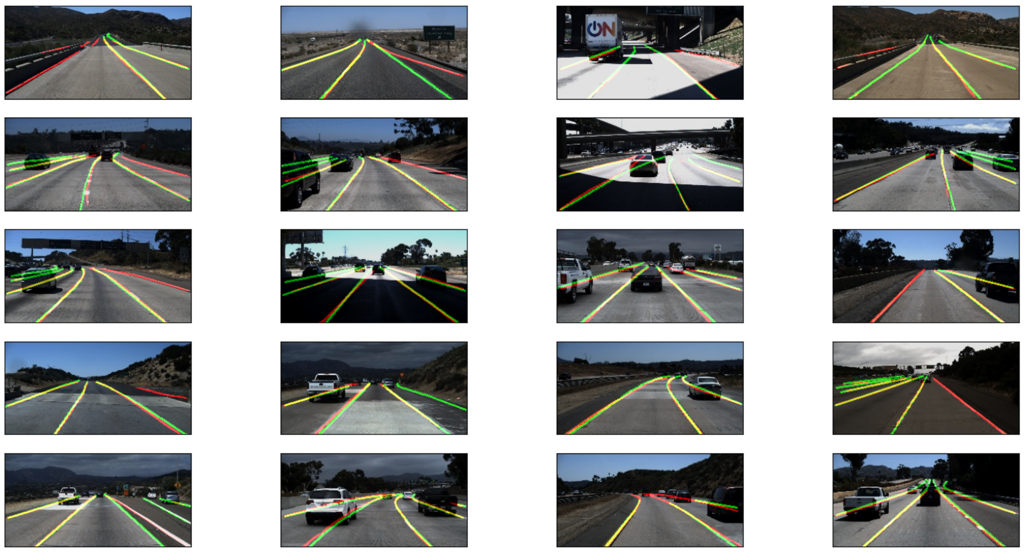

FIGURE 4.11: The 20 predicted scenarios with the worst AUPRC calculated by UNet-LSTM.

4.7.4 Running time

Until now we have analyzed the system just in terms of ROC, PRC, accuracy and F1, but these are not the only possible metrics. For a system like this, which might run embedded, profile and real time capability is of vital importance. The fact, that the proposed architecture takes a sequence of frames as inputs and additionally adds ConvLSTM layer between the encoder and decoder, might have an influence on the running time. Of course, to process a sequence of input images more time is required than just to process a single one, like LaneNet, UNet or SegNet do. However, theoretically the encoder just need to process the actual frame, because the n-last frame have been already procedure and the feature maps stored in memory. Even the n-input frames can be encoded simultaneously/parallelly. Therefore running time can be reduce at the cost of memory or GPU FLOPS. The measured inference time parallelizing the encoder is presented in Table 4.6.

Network	**Average inference time [ms]**
LaneNet	26.512
LaneNet-LSTM*	51.317
UNet	**9.691**
UNet-LSTM	53.728
SegNet	22.463
SegNet-LSTM	63.682

TABLE 4.6: Average inference time. * refers to LaneNet-LSTM with 32 filters ConvLSTM2D and dropout of 0.1.

Until now we have evaluated the networks in terms of space complexity, this means how many parameters they have or how much memory we need to store them. But, as demonstrated here, space complexity and time complexity are not necessarily correlated. For example, SegNet has 24 times more trainable parameters than LaneNet, but it executes approximately 15% faster.

Chapter 5

Discussion

The results of the proposed architecture LaneNet-LSTM can be analyzed/discussed in terms of three major metrics:

1. **Performance**: the proposed network achieves state-of-the-art performance on the TuSimple dataset. Under challenging situations, it shows a quantitative improvement of 4.8% in AUPRC in comparison with single-frame architectures (LaneNet), without outperforming though (by 2.9% in AUPRC) other deeper/bigger multiple-frame architectures like UNet-LSTM. One key conclusion is that most of the well established datasets in the field of autonomous driving just consider "ideal" conditions, and although two different architectures could show similar performance under those, the long tale influence has to definitely be taken into account. In the analyzed case, the performance of single-frame networks drops more abruptly than multi-frame architectures under challenging situations, leading to the conclusion that temporal information help under these adverse situations and validating the hypothesis formulated in section 1.3.

2. **Number of parameters**: due to the fact that the ultimate goal is that these networks run on embedded systems with limited resources and under real time conditions, other metrics rather than performance, like the number of parameters and running time, have to be taken into account. Although the proposed architecture is not as light as a single-frame network, like LaneNet (it has approximately 1.8 times more trainable parameters), it is lighter than other available multi-frames architectures like UNet-LSTM (approximately 80 times less trainable parameters). Another important fact that it should be also mentioned is that the proposed architecture, although it is much smaller, works with higher image resolutions (512x288 pixels against 256x144) leading to smaller segmentation granularity.

3. **Running time**: although the running time is not exactly the same as the number of parameters, they are usually directly correlated. Systems based on single-frame networks, like LaneNet, need approximately half of the time than multi-frame architectures as the one proposed in this work or the UNet-LSTM [59]. One of the advantages of the multiple-frame architecture is that encoder calculations can be

easily parallelized or the result from previous frames saved on the memory, saving computational time at cost of GPU FLOPS or memory.

Chapter 6

Conclusion

In this work, a new variant of a CNN and RNN network was proposed based on already available researches. This architecture is based on a encoder/decoder framework, which takes multiple frames as input and predicts the lane on the last one at pixel level (semantic segmentation). The CNN encoder extracts the feature maps from the different frames, which serve as input to the ConvLSTM. Finally the CNN decoder is responsible for the reconstruction and prediction itself.

The proposed architecture achieves state-of-the-art results on TuSimple dataset, outperforming single-frame networks on challenging situations (shadows, entry and exit of tunnels, high contrasts, etc.) and only overcome by other deeper/bigger multi-frame systems working at lower resolutions [59]. In terms of memory, this fact makes the proposed system more suitable for embedded applications.

In a future work, other types of encoder/decoder architectures or RNN can be investigated. This research has mainly focused on the segmentation part and no so much on the clustering part. For future work, further investigation in this direction might be needed. Another important aspect is that in this work we have focused on 2D, but this information has to be transformed to 3D in order to be used for the motion planning. This is a step that might be also investigated in further projects.

Acknowledgment: this work was supported by AVL Software and Functions GmbH in Regensburg, Germany.

Bibliography

[1] URL: https://paperswithcode.com/sota/lane-detection-on-tusimple (visited on 03/07/2021).

[2] URL: https://github.com/TuSimple/tusimple-benchmark/tree/master/doc/lane_detection (visited on 03/07/2021).

[3] URL: https://www.tensorflow.org/ (visited on 03/13/2021).

[4] URL: https://keras.io/ (visited on 03/13/2021).

[5] URL: http://karpathy.github.io/2019/04/25/recipe/ (visited on 03/13/2021).

[6] Mohamed Aly. "Real time Detection of Lane Markers in Urban Streets". In: *CoRR* abs/1411.7113 (2014). arXiv: 1411.7113. URL: http://arxiv.org/abs/1411.7113.

[7] A. A. Assidiq et al. "Real time lane detection for autonomous vehicles". In: (2008), pp. 82–88. DOI: 10.1109/ICCCE.2008.4580573.

[8] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation". In: *CoRR* abs/1511.00561 (2015). arXiv: 1511.00561. URL: http://arxiv.org/abs/1511.00561.

[9] A. Borkar, M. Hayes, and M. T. Smith. "Robust lane detection and tracking with ransac and Kalman filter". In: (2009), pp. 3261–3264. DOI: 10.1109/ICIP.2009.5413980.

[10] Paula Branco, Luís Torgo, and Rita P. Ribeiro. "A Survey of Predictive Modelling under Imbalanced Distributions". In: *CoRR* abs/1505.01658 (2015). arXiv: 1505.01658. URL: http://arxiv.org/abs/1505.01658.

[11] T. Bruls et al. "Mark Yourself: Road Marking Segmentation via Weakly-Supervised Annotations from Multimodal Data". In: (2018), pp. 1863–1870. DOI: 10.1109/ICRA.2018.8460952.

[12] C. Caraffi, S. Cattani, and P. Grisleri. "Off-Road Path and Obstacle Detection Using Decision Networks and Stereo Vision". In: *IEEE Transactions on Intelligent Transportation Systems* 8.4 (2007), pp. 607–618. DOI: 10.1109/TITS.2007.908583.

[13] A. F. Cela et al. "Lanes Detection Based on Unsupervised and Adaptive Classifier". In: (2013), pp. 228–233. DOI: 10.1109/CICSYN.2013.40.

[14] H. Choi and S. Oh. "Illumination invariant lane color recognition by using road color reference neural networks". In: (2010), pp. 1–5. DOI: 10.1109/IJCNN.2010.5596304.

[15] Junyoung Chung et al. "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling". In: *CoRR* abs/1412.3555 (2014). arXiv: 1412.3555. URL: http://arxiv.org/abs/1412.3555.

[16] F. Coşkun et al. "Real time lane detection and tracking system evaluated in a hardware-in-the-loop simulator". In: (2010), pp. 1336–1343. DOI: 10.1109/ITSC.2010.5625111.

[17] Juan Collado et al. "Adaptative Road Lanes Detection and Classification". In: *Lecture Notes in Computer Science* 4179 (Sept. 2006). DOI: 10.1007/11864349_105.

[18] R. Danescu and S. Nedevschi. "Probabilistic Lane Tracking in Difficult Road Scenarios Using Stereovision". In: *IEEE Transactions on Intelligent Transportation Systems* 10.2 (2009), pp. 272–282. DOI: 10.1109/TITS.2009.2018328.

[19] Richard O. Duda and Peter E. Hart. "Use of the Hough Transformation to Detect Lines and Curves in Pictures". In: *Commun. ACM* 15.1 (Jan. 1972), 11–15. ISSN: 0001-0782. DOI: 10.1145/361237.361242. URL: https://doi.org/10.1145/361237.361242.

[20] F. A. Gers, J. Schmidhuber, and F. Cummins. "Learning to forget: continual prediction with LSTM". In: 2 (1999), 850–855 vol.2. DOI: 10.1049/cp:19991218.

[21] Mohsen Ghafoorian et al. "EL-GAN: Embedding Loss Driven Generative Adversarial Networks for Lane Detection". In: *CoRR* abs/1806.05525 (2018). arXiv: 1806.05525. URL: http://arxiv.org/abs/1806.05525.

[22] Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. Cambridge, MA, USA: MIT Press, 2016.

[23] Ian J. Goodfellow et al. "Generative Adversarial Networks". In: (2014). arXiv: 1406.2661 [stat.ML].

[24] Robert M. Haralick and Linda G. Shapiro. *Computer and Robot Vision*. 1st. USA: Addison-Wesley Longman Publishing Co., Inc., 1992. ISBN: 0201569434.

[25] Alberto Fernández Hilario et al. *Learning from Imbalanced Data Sets*. Springer International Publishing, 2018. ISBN: 978-3-319-98074-4.

[26] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[27] Yuenan Hou et al. "Learning Lightweight Lane Detection CNNs by Self Attention Distillation". In: *CoRR* abs/1908.00821 (2019). arXiv: 1908.00821. URL: http://arxiv.org/abs/1908.00821.

[28] Yen-Chang Hsu et al. "Learning to Cluster for Proposal-Free Instance Segmentation". In: *CoRR* abs/1803.06459 (2018). arXiv: 1803.06459. URL: http://arxiv.org/abs/1803.06459.

[29] David H. Hubel and Torsten N. Wiesel. In: ().

[30] Claudio Jung and Christian Kelber. "Lane following and lane departure using a linear-parabolic model". In: *Image and Vision Computing* 23(13) (Nov. 2005), pp. 1192–1202. DOI: 10.1016/j.imavis.2005.07.018.

[31] Seokwoo Jung et al. "Towards Lightweight Lane Detection by Optimizing Spatial Embedding". In: (2020). arXiv: 2008.08311 [cs.CV].

[32] Yeongmin Ko et al. "Key Points Estimation and Point Instance Segmentation Approach for Lane Detection". In: (2020). arXiv: 2002.06604 [cs.CV].

[33] C. Kreucher and S. Lakshmanan. "A frequency domain approach to lane detection in roadway images". In: 2 (1999), 31–35 vol.2. DOI: 10.1109/ICIP.1999.822849.

[34] C. Kreucher and S. Lakshmanan. "LANA: a lane extraction algorithm that uses frequency domain features". In: *IEEE Transactions on Robotics and Automation* 15.2 (1999), pp. 343–350. DOI: 10.1109/70.760356.

[35] Y LeCun. "Generalization and network design strategies". In: *Technical Report CRG-TR-89-4, University of Toronto* (1989).

[36] J. Li et al. "Deep Neural Network for Structural Prediction and Lane Detection in Traffic Scene". In: *IEEE Transactions on Neural Networks and Learning Systems* 28.3 (2017), pp. 690–703. DOI: 10.1109/TNNLS.2016.2522428.

[37] Xiangyang Li et al. "Lane Detection and Tracking Using a Parallel-snake Approach". In: *Journal of Intelligent and Robotic Systems* 77 (Mar. 2014). DOI: 10.1007/s10846-014-0075-0.

[38] Q. Lin, Y. Han, and H. Hahn. "Real-Time Lane Departure Detection Based on Extended Edge-Linking Algorithm". In: (2010), pp. 725–730. DOI: 10.1109/ICCRD.2010.166.

[39] Andre Linarth and Elli Angelopoulou. "On feature templates for Particle Filter based lane detection". In: *Conference Record - IEEE Conference on Intelligent Transportation Systems* (Oct. 2011), pp. 1721–1726. DOI: 10.1109/ITSC.2011.6083016.

[40] H. Loose, U. Franke, and C. Stiller. "Kalman Particle Filter for lane recognition on rural roads". In: (2009), pp. 60–65. DOI: 10.1109/IVS.2009.5164253.

[41] Abdelhamid Mammeri, Azzedine Boukerche, and Guangqian Lu. "Lane detection and tracking system based on the MSER algorithm, hough transform and kalman filter". In: (Sept. 2014). DOI: 10.1145/2641798.2641807.

[42] Luca Mora, Xinyi Wu, and Anastasia Panori. "Mind the gap: Developments in autonomous driving research and the sustainability challenge". In: *Journal of Cleaner Production* 275 (2020), p. 124087. ISSN: 0959-6526. DOI: https://doi.org/10.1016/j.jclepro.2020.124087. URL: https://www.sciencedirect.com/science/article/pii/S0959652620341329.

[43] Davy Neven et al. "Towards End-to-End Lane Detection: an Instance Segmentation Approach". In: *CoRR* abs/1802.05591 (2018). arXiv: 1802.05591. URL: http://arxiv.org/abs/1802.05591.

[44] Xingang Pan et al. "Spatial As Deep: Spatial CNN for Traffic Scene Understanding". In: *CoRR* abs/1712.06080 (2017). arXiv: 1712.06080. URL: http://arxiv.org/abs/1712.06080.

[45] Russell D. Reed and Robert J. Marks. *Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks*. MIT Press, 1999.

[46] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: *CoRR* abs/1505.04597 (2015). arXiv: 1505.04597. URL: http://arxiv.org/abs/1505.04597.

[47] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning Representations by Back-propagating Errors". In: *Nature* 323.6088 (1986), pp. 533–536. DOI: 10.1038/323533a0. URL: http://www.nature.com/articles/323533a0.

[48] M. A. Selver et al. "Camera based driver support system for rail extraction using 2-D Gabor wavelet decompositions and morphological analysis". In: (2016), pp. 270–275. DOI: 10.1109/ICIRT.2016.7588744.

[49] Xingjian Shi et al. "Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting". In: *CoRR* abs/1506.04214 (2015). arXiv: 1506.04214. URL: http://arxiv.org/abs/1506.04214.

[50] Connor Shorten and Taghi M. Khoshgoftaar. "A survey on Image Data Augmentation for Deep Learning". In: abs/1505.01658 (2019).

[51] U. Suddamalla et al. "A novel algorithm of lane detection addressing varied scenarios of curved and dashed lanemarks". In: (2015), pp. 87–92. DOI: 10.1109/IPTA.2015.7367103.

[52] T. Suttorp and T. Bucher. "Learning of Kalman Filter Parameters for Lane Detection". In: (2006), pp. 552–557. DOI: 10.1109/IVS.2006.1689686.

[53] W. Wang et al. "A Learning-Based Approach for Lane Departure Warning Systems With a Personalized Driver Model". In: *IEEE Transactions on Vehicular Technology* 67.10 (2018), pp. 9145–9157. DOI: 10.1109/TVT.2018.2854406.

[54] Y. Wang, D. Shen, and E. Teoh. "Lane detection using Catmull-Rom splice". In: (1998).

[55] Y. Xing et al. "Advances in Vision-Based Lane Detection: Algorithms, Integration, Assessment, and Perspectives on ACP-Based Parallel Vision". In: *IEEE/CAA Journal of Automatica Sinica* 5.3 (2018), pp. 645–661. DOI: 10.1109/JAS.2018.7511063.

[56] Yinghua He, Hong Wang, and Bo Zhang. "Color-based road detection in urban traffic scenes". In: *IEEE Transactions on Intelligent Transportation Systems* 5.4 (2004), pp. 309–318. DOI: 10.1109/TITS.2004.838221.

[57] Yue Wang, Eam Khwang Teoh, and Dinggang Shen. "Lane detection using B-snake". In: (1999), pp. 438–443. DOI: 10.1109/ICIIS.1999.810313.

[58] S. Zhou et al. "A novel lane detection based on geometrical model and Gabor filter". In: (2010), pp. 59–64. DOI: 10.1109/IVS.2010.5548087.

[59] Qin Zou et al. "Robust Lane Detection from Continuous Driving Scenes Using Deep Neural Networks". In: *CoRR* abs/1903.02193 (2019). arXiv: 1903.02193. URL: http://arxiv.org/abs/1903.02193.

Appendix A

Student's t-test

1. **Formulate the hypothesis**: the null hypothesis $H_0 : \mu_1 = \mu_2$ states that both methods have the same mean and the alternative hypothesis $H_1 : \mu_1 \neq \mu_2$ states that one is statistically better then the other. μ_1 and μ_2 refers to the means of the two populations from which the samples were drawn.

2. **Calculate test statistic**: the next step is calculate the test statistics

$$t_{obs} = \frac{(\overline{x}_1 - \overline{x}_2) - (\mu_1 - \mu_2)}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}} := \frac{(\overline{x}_1 - \overline{x}_2)}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}} \tag{A.1}$$

   where:
   $\overline{x}$: drawn samples mean.
   μ: population mean. According to the null hypothesis $H_0 : \mu_1 = \mu_2$. This means that $\mu_1 - \mu_2 = 0$.
   s^2: standard deviation of the drawn samples.
   n: number of drawn samples.

3. **Calculate critical value**: finally the critical value $-t_{\alpha,\nu}$ can be calculated.

$$\nu = \frac{(\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2})^2}{\frac{(\frac{s_1^2}{n_1})^2}{n_1-1} + \frac{(\frac{s_2^2}{n_2})^2}{n_2-1}} \tag{A.2}$$

   So with this information and using a significance level of 5% we can obtain $t_{0.05,\nu}$.

4. **Conclusion**: the rejection region is from $-\infty$ to $t_{0.05,\nu}$ and from $t_{0.05,\nu}$ to $+\infty$. If we calculate this procedure for the three best analyzed methods, the test statistic t_{obs} is outside the calculated rejection regions so we do not reject the null hypothesis H_0. This means that in terms of the initial question, at 5% significance level, we do not reject the hypothesis that the mean of the two populations is the same, or there is no sufficient evidence in the data to conclude that the two populations are different. The final conclusion is thus, that the three architectures LaneNet, LaneNet-LSTM and UNet-LSTM have statistically the same performance.

Comparison	t_{obs}	ν	$t_{0.05,\nu}$	$H_0 : \mu_1 = \mu_2$
LaneNet vs LaneNet-LSTM	-0.253	7.983	-1.851	Accepted
LaneNet vs UNet-LSTM	-0.127	7.650	-1.860	Accepted
LaneNet-LSTM vs UNet-LSTM	0.101	7.778	-1.866	Accepted

TABLE A.1: Results of Student's t-test for the comparison between LaneNet, LaneNet-LSTM and UNet-LSTM.

Appendix B

Challenging conditions on the TuSimple dataset

TuSimple dataset is recorded exclusively under good weather conditions, which of course represents just a very small subset of all the possible scenarios under which autonomous driving shall be able to work. The main goal of the proposed methods is tackle the problem of challenging situations like shadows, light reflections, lane occlusion, etc. To calculate the performance under these circumstances the validation set has been manually labeled to find the more difficult situations: shadows, entry and exit of tunnels, high contrasts, etc. Some examples of these frames are presented in Figure B.1.

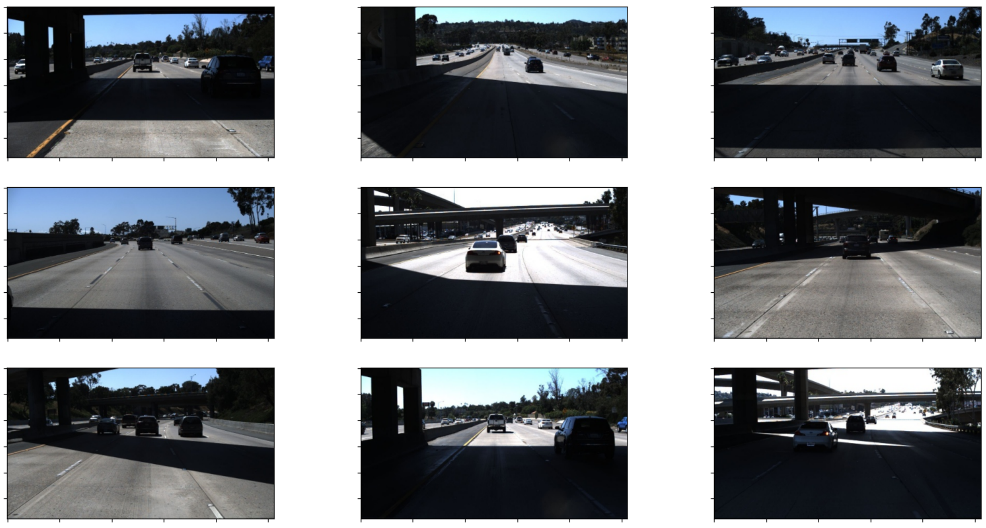
FIGURE B.1: Some of the frames categorized as challenging in the TuSimple validation dataset.

In total the TuSimple validation dataset contains only 30 images under those circumstances, which it might be argued that the calculated results might not be statistically relevant.