CS230

Traffic Lane Detection with Fully Convolutional Networks

Shengchang Zhang, Ahmed EI Koubia, Khaled Abdul Karim Mohammed
(victorzh, Akoutbia, Khaledm)@stanford.edu

Abstract

Automatic lane detection is a critical technology that enables self-driving cars to properly position themselves in a multi-lane urban driving environments. However, detecting diverse road markings in various weather conditions is a challenging task for conventional image processing or computer vision techniques. In recent years, the application of Deep Learning and Neural Networks in this area has proven to be very effective. In this project, we designed an Encoder-Decoder, Fully Convolutional Network for lane detection. This model was applied to a real-world large scale dataset and achieved a high level of accuracy that outperformed our baseline model.

KEYWORDS: Lane Detection, Fully Convolutional Network, Dice-coefficient Loss, Semantic Segmentation

1 Introduction

Lane detection is one of the key techniques that enables Advanced Driver Assistance System(ADAS) and autonomous driving systems to identify lanes on the road. It provides the accurate location and shape of each lane. The lack of distinctive features can cause lane detection algorithms to be confused by other objects with similar appearance. Moreover, the inconsistent number of lanes on a road, as well as diverse lane line patterns, e.g. solid, broken, single, double, merging, and splitting lines, further hampers performance.

We started with the initial idea of applying instance segmentation to differentiate between lanes on the road, and label them with different colors. As we moved forward, we discovered that with our existing baseline model, we were not getting good results in terms of detecting specific lane instances. Based on the discussion we had with our instructor, we decided to apply semantic segmentation in solving our lane detection problem.

2 Related Work

In recent years, many sophisticated lane detection methods have been proposed:

Dataset processing: Qin Z., etc.[1]. investigated lane detection by using multiple frames from a continuous driving scene, and proposed a hybrid deep learning architecture, which combines a Convolutional Neural Network (CNN) and a Recurrent Neural Network (RNN). Han Ma, etc.,[2], proposed a multi-lane detection algorithm developed based on an optimized dense disparity map estimation. However, they do not perform well in challenging and dim environments. This is particularly true when strong interference such as crossings and turnings exist. In his paper[3], Davy Neven etc. attempted to define the lane detection problem as an instance segmentation problem, where each lane forms its own instance that can be trained end-to-end. To parameterize the segmented lane instances, he proposed to apply a learned perspective transformation, conditioned on the image, in contrast to a fixed ”bird’s eye view” transformation.

Models and Algorithms: Ze Wang,[4] proposed LaneNet, a deep neural network based method to break down the lane detection into two stages: lane-edge proposal and lane-line localization. However, their model detects lane lines only, and performs detections on similar lane marks on the road like arrows and characters. Global Convolution Networks (GCN)[5] model used color-based segmentation to address both classification and localization issues. They offer a real-time video transfer system, which requires a great deal of processing power. For ADAS applications, P.R.Chen[7] proposed a Lane Marking Detector(LMD) using a deep CNN to extract robust lane marking features by adopting dilated convolution with shallower and thinner structure to decrease the computational cost.

Code link: https://github.com/khaledmohammed/CULaneDetection ,

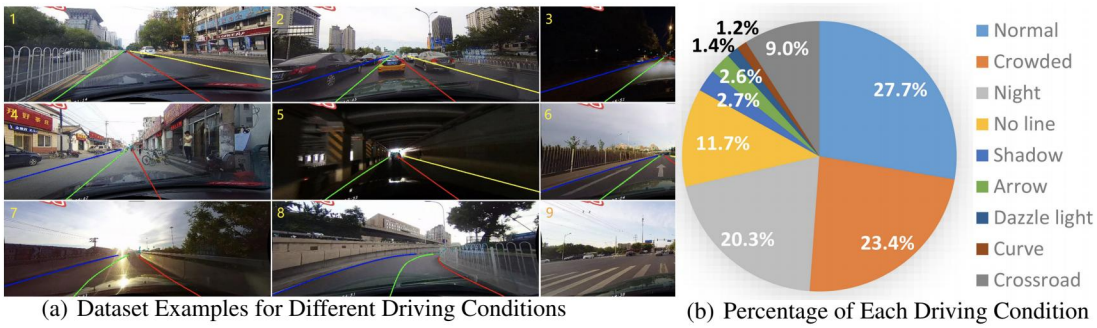
Several state-of-the-art approaches have achieved excellent performance in real applications. However, most methods focused on detecting the lane from one single image, and often lead to unsatisfactory performance in handling extremely challenging situations such as heavy shadow, severe mark degradation and serious vehicle occlusions.

3 Dataset Selection and Description

We had an initial plan to develop our own dataset by mounting cameras on our cars, and possibly drive in different weather and road conditions. As we explored several data collection approaches for our project, we soon realized that creating from scratch our own dataset for lane detection would a very time-consuming process. Instead, we decided to focus the bulk of our efforts on building the Neural Networks model for this project, and use one of the existing online datasets. After an extensive search, we identified two datasets for traffic lane detection: CULane[8] and TuSimple[9]. We concluded that CULane dataset was more suitable for our project because of the way it was structured and documented. The following section provides a detailed description of this dataset.

3.1 CULane Dataset

CULane is a large dataset for lane detection collected by cameras mounted on six different vehicles on the streets of Beijing, China. More than 55 hours of videos were collected and 133,235 frames were extracted. These images were not sorted and have a resolution of 1640 x 590 pixels. The dataset is divided into 88,880 image for training, 9,675 for validation and 34,680 images for test. Figure3.1 provides a breakdown of 9 categories representing different driving conditions captured in this dataset. Challenging driving conditions represent more than 72.3%. Each frame is manually annotated with cubic splices. In case the traffic lanes are occluded by vehicles, the lanes are annotated based on the context. Lanes on the other side of the road are not annotated. Each image has a corresponding annotation text file providing to the x and y coordinates for key lane marking points.



(a) Dataset Examples for Different Driving Conditions (b) Percentage of Each Driving Condition

Figure 1: CULane Dataset

CULane dataset is comprised of two types of images. The first consists of the original images; whereas the second category consists of mask images, which contain only the lane markings based on the original image. In case there are no lane markings, the mask image is empty. Each mask image contains a number representing lane markings from the original image. This number for the mask pixel denotes whether a lane is present on that particular pixel, and it takes 5 possible values. When a mask pixel has a zero value, it means a lane is absent. On the other hand, if a mask pixel has a value between 1 and 4, this implies that a lane is present on this pixel and the number denotes the lane number. In short, every mask image may have from 0 to 4 lanes detected, and the lanes are numbered from left to right.

3.2 Train and Dev Datasets

We developed a Python module that scans the dataset folders dynamically to collect the image and label file names. This enabled us to increase or decrease the amount of data for training without any code modifications. Furthermore, we shuffled and divided the dataset into Train (90%) and Dev (10%).

4 Methods

4.1 Data Pre-processing

Initially, we developed a python script to process the dataset, and created serialized Python arrays nested in a list as expected by the baseline model, MLND-Capstone [10]. This list contained the images as Python arrays which were serialized using the Pickle API. We soon identified some challenges with our original pre-processing script. In particular, when we tried to load more than 600 images for training. When all 600 training images were loaded in memory, we started running out of memory.

As a result, we developed a custom Python Generator script for our dataset, which enabled us to load all training images without experiencing any memory issues. To enhance the performance further, we started loading the data in mini-batches, while configuring the optimal mini-batch size based on the GPU memory resources available.

To speed up the training of our algorithm, we reduced the size of our images to one-fifth of their original size while maintaining the same aspect ratio. This enabled us to increase the batch size from 32 to 128, and significantly reduce the training time. As a comparison, LaneNet[3]'s batch size was 4 images for 8 GB GPU. In our case, we can train our model on 8x more images for the same amount of time.

We also modified the training labels to have binary values (i.e. 0 or 1) instead of the 5 values described in section 3.1. By doing so, we have changed the focus of our Network from an instance segmentation to a semantic segmentation problem.

We also experimented with Random Channel shift as a data augmentation technique. However, we have not seen any performance benefits resulting from applying it. Considering that CUlane is very large dataset and captures most of the driving conditions that we were interested in, we didn't see any need for data augmentation in this project.

4.2 Network Architecture

Our model has an Encoder-Decoder Network architecture(Figure 2). The Encoder component consists of 7 Convolutional layers mixed with Maxpooling layers and Dropout regularization. The purpose of the Encoder component is to reduce the size of the the image while capturing its key features into more channels. One the other hand, the Decorder component consists of 6 Deconv or ConvTranspose layers. It up-samples the feature vectors to an image mask, which has same height and width as the original image, on a gray-scale format. The main idea behind choosing this network architecture is to speed up training when processing CUNet higher resolution images. Applying alternative architectures that rely on Convolutinal Layers followed by Fully Connected Layers, would require training very large number of parameters. This could significantly reduce the performance of our neural network.

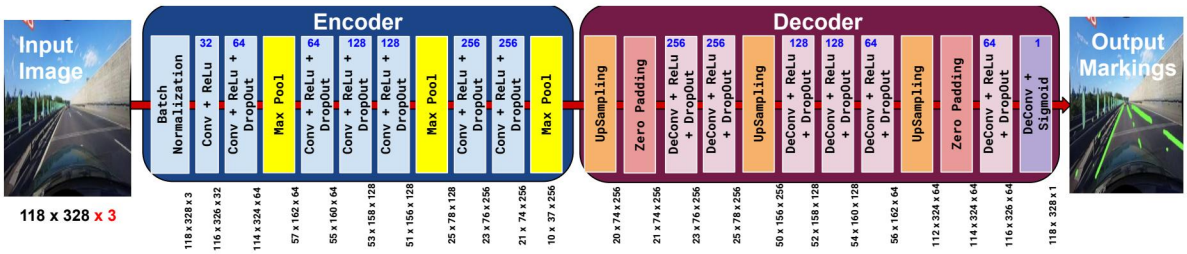


Figure 2: LaneDetect Fully Convolutional Network Architecture

4.3 Baseline Model Selection

We selected MLND-Capstone by Michael Virgo[10] as the baseline model for our project because it provides good documentation and the code is easy to understand. According to Michael's project report, he was able to achieve a good performance on his dataset after only 10 epochs.

4.4 Model Adjustments

CULane images have a different aspect ratio compare to the baseline model's expected input resolution. To accommodate our image dimensions, we had to add 2 ZeroPadding layers to force the model to generate the same dimensions as those of label images. The baseline model used very few filters to capture the features of the images. This may have been sufficient for the simple two-way lane dataset, captured and labeled for the baseline model. To achieve a good performance for CULane dataset, we had to increase the number of filters by 4x for each layer.

4.5 Loss Functions

The baseline model applied Keras' built-in Mean Squared Error(MSE) loss function. Since our labels are binary (i.e. 0 when a pixel is not part of a lane; and 1 when a pixel is part of a lane), it would be appropriate to use the Binary Cross Entropy(BCE) loss function.

However, we didn't observe any performance improvements resulting from the application of the BCE loss function. After careful analysis, we realized that pixels, which are part of lanes markings represent a very small percentage compare to the total number of pixels in an image. Therefore, the labels with the value of 1 are far less than the labels with the value of 0. In calculating the accuracy, MSE and BCE loss functions gave much bigger importance to the dominant label (i.e. label '0').

After some research, we discovered the Dice-coefficient loss function, which provides more importance to the accuracy of label '1' (i.e the location of label '1' in predicted mask should match the location of label '1' in the ground truth). This also ensured that label '0' was in the right location. As a result, we applied the Dice-coefficient cost function(Equation1) to measure the similarity between the actual and predicted labels.

$$DiceLoss(p, \hat{p}) = 1 - \frac{2\langle p, \hat{p} \rangle}{||p||_2^2 + ||\hat{p}||_2^2} \tag{1}$$

5 Experiment and Results

5.1 Hyper-parameter Tuning

We selected the Adam Optimizer during the training of our model. We started with default learning rate of 0.01, and gradually decreased it until it reached a value of 0.0001. We noticed that Adam Optimizer doesn't reduce the Dice Loss if the learning rates were large.

Moreover, we selected a mini-batch size of 128 image and labels which was constrained by our GPU memory. With a small learning rate we trained our model for 600 epochs to provide sufficient time to the optimizer to reduce the loss. The following Figure 3 presents our performance metrics from our training dataset.

5.2 Evaluation Metrics

When we calculated the accuracy of our model, we obtained a value of 99%. However, in a given image, the proportion of TN(pixels not part a lane) is much greater than TP(pixels part of a lane). Therefore, we needed a more reliable performance metric. In this case, we used the F1 score, which enabled us to measure the proportion of predicted correct classifications(TP) from the observations that are actually positive(TP + FN), or predicted to be positive(TP + FP). When we trained our model for 600 epochs, we obtained an F1 score of 85%. Figure 3, shows the cost monotonically decreasing as a function of the number of training iterations; the accuracy at the level of 99% regardless of the number of iterations; more importantly the F1 score initially increasing rapidly during the first 100 iterations, and then stabilizing around 500 iterations.

Moreover, we needed to understand how our F1 score metric performed as a function of several probability thresholds of our classifier. We selected 8 possible values for the probability threshold and calculated their corresponding F1 score. In order to obtain an unbiased estimate of the F1 score, we calculated the average F1 score accross all the 8 probability thresholds.

Figure 4 show that the F1 score reaches a maximum value at a probability threshold of 0.5.

Figure 3: Training Loss, Accuracy and F1 Score

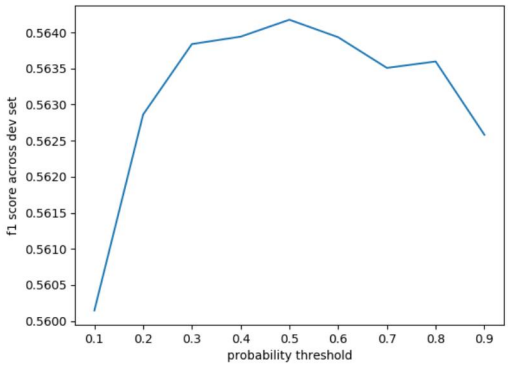


Figure 4: F1 Score vs 8 Probability Thresholds

5.3 Predictions & Results

We compared the predictions of two different models trained with with three different loss functions (Dice Coefficient , MSE and MSE HeatMap) against the Ground Truth. Fig.5 provides a visualization of the results. The first row or example corresponds to a test image with four lanes where the markings are clear; whereas the second row represents a test image with three lanes, in which the lane markings are not clear. We can easily notice that in the first example, both Dice loss function model provided a better prediction of the lanes markings. On the other hand, all models didn't perform very well in predicting the lane marking in the second test example because they seem to be having trouble identifying the lane markings when there are cross lanes and object occlusions.

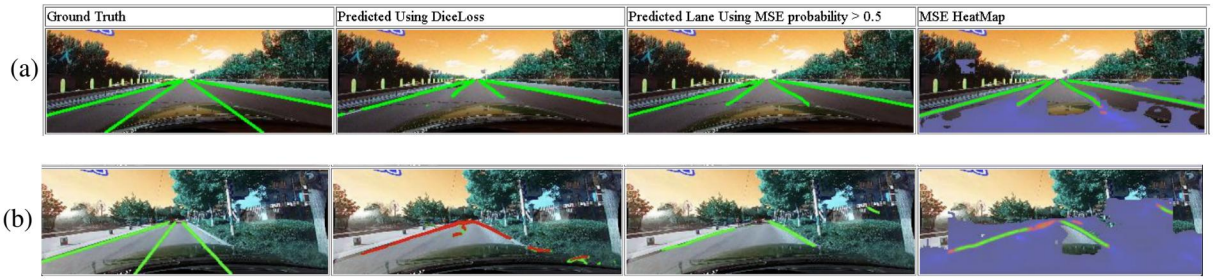


Figure 5: Lane markings results

6 Conclusion/Future Work

In this project we attempted to tackle the complex problem of muti-lane detection using a large-scale and challenging dataset. We extended the baseline model which was designed initially to handle small datasets. We developed custom Python modules to load the dataset efficiently. We tuned the hyper-parameters, extended the convolutional layers, added pooling layers and replaced the MSE cost function with the Dice-coefficient cost function. This enable us to achieve a high accuracy in lane detection.

Further development of this project would include multi-lane and instance detection. In other words, for each image in the CUDataset dataset, we need to identify the numbers of lanes, their location and their unique instance.

To achieve this goals, we believe that it is worthwhile to pursue the following ideas in the future: 1)Instead of training our neural network from scratch we will explore more deeper FCN as a backbone architecture(Alexnet, VGG, and Resnet). This will seed up the training of our model. 2)Explore various Region-Based Convolutional Neural Networks (R-CNN), which are designed for object detection and instance segmentation. This includes Faster-RCNN and Mask-RCNN. 3)To evaluate the result of the model, we need to apply a multi-task loss function to our multi-lane detection model. This include the loss of classification, localization and segmentation mask.

7 Contributions

- Victorzh lead the initial research and evaluation of various Neural Network Models including instance and semantic segmentation. He also worked on finding and selecting the appropriate lane-detection datasets for our project. Moreover, Victor reviewed and compared several research papers about lane-detection and neural networks.
- Khaled worked on various data pre-processing techniques for this project, implemented custom loss functions and evaluation metrics using Keras. He also worked on tuning of the model, analyzed the data generated from various experiments, and generated heat-maps comparing the performance of various models with the Ground Truth.
- Ahmed identified and proposed the code baseline of this project. He also provided a detailed CUlane dataset description, and worked on the evaluation metrics. Moreover, he researched and documented several neural network architectures for this project including RCNN, Fast CNN, Faster CNN and Mask R-CNN. Finally, Ahmed was instrumental in setting up and administering the AWS GPU Instance.

References

[1] Qin Zou, Hanwen Jiang, Qiyu Dai, Yuanhao Yue, Long Chen, Qian Wang. Robust Lane Detection from Continuous Driving Scenes Using Deep Neural Networks. arxiv 2019. arXiv:1903.02193

[2] Han Ma, Yixin Ma, Jianhao Jiao, M Usman Maqbool Bhutta. Multiple Lane Detection Algorithm Based on Optimised Dense Disparity Map Estimation. arXiv:1808.09128v1 [cs.CV] 28 Aug 2018

[3] D. Neven, B. D. Brabandere, S. Georgoulis, M. Proesmans, and L. V.Gool, "Towards end-to-end lane detection: an instance segmentation approach," CoRR, vol. abs/1802.05591, 2018.

[4] Ze Wang,Weiqiang Ren, Qiang Qiu. LaneNet: Real-Time Lane Detection Networks for Autonomous Driving. arXiv:1807.01726v1 [cs.CV] 4 Jul 2018

[5] Mahale Chaoran Chen and Wenhui Zhang * equal contribution. End to End Video Segmentation for Driving : Lane Detection For Autonomous Car. arXiv:1812.05914v1 [cs.CV] 13 Dec 2018

[6] Mohsen Ghafoorian, Cedric Nugteren,Nora Baka, Olaf Booij, Michael Hofmann. EL-GAN: Embedding Loss Driven Generative Adversarial Networks for Lane Detection. arXiv:1806.05525v2 Jul 2018

[7] Ping-Rong Chen*, Shao-Yuan Lo*, Hsueh-Ming Hang National Chiao Efficient Road Lane Marking Detection with Deep Learning.. arXiv:1809.03994v1

[8] CULane: `https://drive.google.com/drive/folders/1mSLgwVTiaUMAb4AVOWwlCD5JcWdrwpvu`

[9] Dataset Tusimple: `https://github.com/TuSimple/tusimple-benchmark/wiki`

[10] https://github.com/mvirgo/MLND-Capstone

[11] Vijay Badrinarayanan, Alex Kendall, Roberto Cipolla.SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation. IEEE Transactions on Pattern Analysis and Machine... 2016 DOI:10.1109/TPAMI.2016.2644615

[12] Shriyash Chougule1, Nora Koznek2, Asad Ismail, Ganesh Adam1, Vikram Narayan2 and Matthias Schulze. Reliable multilane detection and classification by utilizing CNN as a regression network. ECCV 2018. Https://link.springer.com/conference/eccv

[13] https://en.wikipedia.org/wiki/S%C3%B8rensen%E2%80%93Dice_coefficient