Lane Detection for Autonomous Vehicles using modified Spatial CNN

Ashar Alam
Stanford University
ashar.alam@stanford.edu

Muhammad Aadil Khan
Stanford University
maadilk@stanford.edu

Abstract

For our project, we were interested in working on lane detection for autonomous vehicles using lane markings painted on the roads. The task of lane detection is still vital to the success of autonomous vehicles because it is pertinent that an autonomous vehicle can exactly recognize which lane it is in and where it needs to go. If it is not able to do that, the vehicle might be going in the middle of two lanes, which can be dangerous for other vehicles in the vicinity. Also, lane detection is inevitable to perform maneuvers like lane changes and predicting other vehicles' behavior. Our main work was based on Xingang et.al, where the authors use Spatial CNNs to identify spatial features in the scene that are relevant to identifying lane markings. In this project, we modified the SCNN architecture to make the model simpler and less memory intensive and tested it on the TuSimple Dataset. The results showed that we achieved similar performance and accuracies with our modified models compared to the baseline SCNN.
You can find our Code Repo here (we have set up a Github within Colab and commit from there): https://github.com/asharalam11/LaneNet/tree/gcp_vm

1. Introduction

As the pursuit of full-self driving intensifies in the autonomous driving community, many are turning to computer vision techniques to ensure that cars can see the world as humans do. Scene understanding has always been a very challenging task, which comprises of tasks like object detection, semantic segmentation etc. Moreover, ensuring safety of vehicles while driving autonomously is of paramount importance. A lot of autonomous driving features being provided in current cars are dubbed as ADAS (Advance Driver Assistance Systems), which include features like Lane Keeping Assist and Adaptive Cruise Control. In such a scenario, reliable lane detection becomes a very important challenge to ensure these functionalities. This task is especially challenging because driving scenarios are subjected to various lighting conditions impacted by changes in weather, times of day or occlusions; which make computer vision based lane detection very difficult. Also, there are inconsistent number of lanes and different lane marking patterns like broken, single, merging, splitting or faded lane lines. Owing to these variations, the task of lane marking detection can be very difficult at intersections; but we can simplify this task to limiting ourselves to freeways as exemplified by the Tusimple Dataset [1], which is based on freeway driving. Our idea is to take the Spatial CNN model suggested in [2] and modify it by varying the convolutional layers to try to simplify the network and also tune the hyperparameters and loss functions to try to achieve similar level of accuracies with a simpler modified model.
Our task was to identify the location of lanes in the image. The input to our model was an image from the dataset, whose size was reduced. We then use a pre-trained backbone followed by an SCNN model to output an image with predictions on every pixel. Thus, output of our model is an image, where each pixel is a value representing the probabilities of existence of lanes in that particular pixel. Then, we extract the position of lanes from the output image predictions based on a threshold. Finally, we compare the predicted lanes against ground truth lane labels to determine the performance accuracy of our model.

2. Related Work

2.1. Scene Understanding and Segmentation

Initial approaches for lane detection combined CNNs with traditional computer vision feature extraction techniques like [3]. Researchers also drew inspiration from hierarchical neural networks to develop multitask deep CNNs to detect the presence of targets and their geometric attributes [4]. With the rapid advancement in computer vision techniques for deep learning, many researchers generated their own dataset and experimented with their own CNN models like Deeplanes [5]. This was followed by Fully Convolutional Neural Network approaches like [6], which used

RANSAC for post-processing.

With semantic segmentation gaining popularity, many researchers shifted their focus to semantic segmentation for lane detection. SegNet [7] uses encoder-decoder architecture to perform semantic segmentation. The encoder network is identical to VGG16 with each encoder layer paired with a decoder layer. The encoder-decoder architecture is followed by a pixel-wise classification layer. [8] uses cascaded CNNs to perform instance segmentation and feature extraction. In this approach, they train a CNN for lane boundary instance segmentation, and then use another CNN to extract a descriptor for each lane boundary and process it. Another approach uses instance segmentation for end-to-end lane detection [9].

An interesting research focussed on using lightweight CNN models using Self Attention Distillation using attention maps to enable the network to perform well even in the presence of occlusions or complex scenarios [10]. With 3D computer vision techniques gaining traction, [11] developed 3D-LaneNet to correctly identify lane markings from sensors onboard the vehicle. There has also been some interesting approaches to Lane Detection by Zhou et.al [12] with encoder-decoder CNNs and LSTMs. In this approach, they use Encoder CNN to encode information from continuous frames of images in a clip and process these as time series data through a convolutional LSTM. Finally, they use a decoder CNN to produce prediction of the lane in the final image frame.

2.2. Lane Detection and Spatial CNN

Pan et al [2] proposed spatial CNN, with the intention of capturing spatial information via a specially designed CNN structure. This approach focuses on learning the spatial relationship and the smooth continuous prior of lane markings. This helps to retain the features like lanes having strong structural priors but less appearance clues. This model won the 1st place in the TuSimple Benchmark Lane Detection challenge but has complex, less intuitive specially designed lane detection layers, which also take more time to train. In our project, we modified this model, by modifying certain layers and employing techniques like increasing dropout to reduce the model complexity in pursuit of simplifying the model while maintaining similar accuracy.

3. Dataset

We used the TuSimple Dataset [1], which contains 4K 1 second-video clips of driving on highway. Each clip has 20 frames of dimensions 1280×720 pixels, with the last ($\mathbf{20^{th}}$) frame labelled in a JSON format. Each image has 4 labelled lane markings (current lane and left/right lanes), (if they exist) with a list of points specifying coordinates in the image pixel space. The dataset is good for this task because it is reasonably complex and contains a wide variety of images ranging from different daytimes to different weather conditions as can be seen in Figure 1. Moreover, the dataset also has images in which the lanes are obstructed as shown in Figure 2 making it harder to predict them. This ensures that once the model is fully trained, it would be versatile enough to detect lane in numerous different conditions.

3.1. Data Preprocessing

We generated a binary label image from the JSON file for each image for visualizing the lanes as seen in Figure 3. We used Python OpenCV library to generate these labels. We also resized each image before feeding it into the Neural Network by resizing it to 512×288, Rotating it by 5 degrees and also Normalizing it to zero-center the images. We calculated the mean and standard deviation for all the images and then normalized the images with these metrics. Instead of manually loading the data, we created a Pytorch DataLoader object to load our dataset so that it can be used seamlessly in the Pytorch framework. We took 4K images and randomly shuffled to train (80%), validation (10%) and test (10%) splits.

Figure 1. Road Scene in Low Light

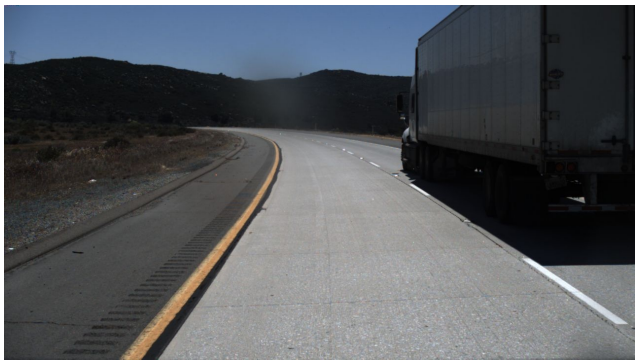

Figure 2. Road Scene with Lane Obstructed

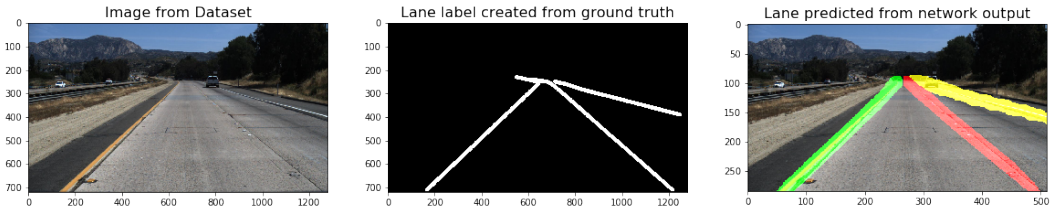

Figure 3. Image from Dataset and its Prediction

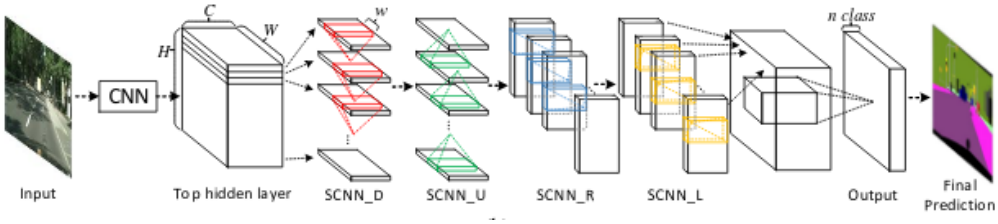

Figure 4. Model architecture of Spatial CNN [1]

4. Methods

4.1. Spatial CNN: Model Architecture

Since, we are using the Spatial CNN idea suggested by Pan et al. [2]; we will be explaining their approach for Spatial CNN. Hence, this section is heavily borrowed from the contents of their paper [2].

CNNs are usually very effective at extracting semantic information from raw pixels like in the tasks of feature detection or object detection. However, they struggle with tasks of identifying the spatial relationships (rotational and translational relationships) or represent the "pose" of features. For example, it is difficult for a CNN to identify lane lines as different types of lane lines have different colors, gaps, occlusions etc. and CNNs are not very effective at identifying lanes just by extracting semantic features.

As noted by Pan et al. [2], traditionally, spatial relationships have been traditionally modeled by Markov Random Fields (MRF) or Conditional Random Fields (CRF). These traditional methods employ normalizing (using Softmax) followed by message passing(channel wise convolutions) followed by 1×1 convolution and softmax all repeated for iterations. The message passing estimators are very effective at performing structured prediction. However, this method is computationally expensive and hard to learn as each pixel receives information from all other pixels. This renders this approach impractical as it is not suitable for real time applications like autonomous driving.

In traditional CNNs, each convolutional layer applies convolution and nonlinear activation to ouput from preceding layer and forwards its output to the next layer. SCNN increases emphasis on spatial information by treating individual row and column feature maps as layers. Then, they facilitate message passing of pixel information between neurons within the same layer by applying the slice process sequentially. Under this sceheme, a slice passes information to the next slice only after it has received information from preceding slices.

The idea behind 'Spatial CNN' is to spatially propagate the spatial relationship in lane markings via the CNNs ahead. Figure 4. shows the Spatial CNN taking a 3-D Tensor of size $C \times H \times W$ as input, which is split into H slices. The first slice is then sent to a convolutional layer with C kernels of size $C \times w$ and the output is added to the next slice to produce a new slice. These slices are further propagated through the next convolutional layers with spatial directions (up, down, left and right directions indicate the direction of slicing). SCNN seems similar to the idea of Recurrent Neural Networks.

Let's try to better understand the slicing scheme as explained by Pan et. al [2]. Let us consider a 3-D Kernel Tensor W with each element $W_{i,j,k}$ (where i is the i^{th} channel of the last slice and j is the j^{th} channel of the current slice, with k as the offset between two elements in channel i and j) denoting the weight between an element in channel i an channel j. For an input tensor element $X_{i,j,k}$ (where i is the channel index, j is the row index and k is the column index), we can write the forward pass of the SCNN as:

$$X'_{i,j,k} = \begin{cases} X_{i,j,k} & j = 1 \\ X_{i,j,k} + f(\sum_m \sum_n & \\ X'_{m,j-1,k+n-1} \times W_{m,i,n}) & j = 2, 3, ..., \end{cases} \quad (1)$$

where f is a nonliner activation function like ReLU and X' denotes updated element. Like Recurrent Neural Network, SCNNs share the kernel weights across all slices in the spirit

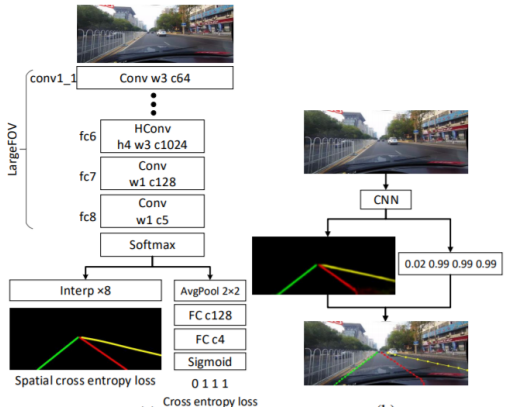

Figure 5. Working of SCNN

of parameter sharing.

4.2. Working of SCNN Model

The task of lane detection involves precise prediction of lane curves using pixel-wise identification. Traditional approaches like [3] train CNN models for lane presence and perform clustering or RANSAC. As seen in Figure 5, [2] Pan et.al treated lane markings as four different classes (first lane is non-existent in this image; colored green, red and yellow). The SCNN model takes the raw image as input and outputs probability maps for each lane marking curve like semantic segmentation tasks. These probability maps are then passed through a small network to predict final cubic splines for lanes.
Moreover, we also get four values of probabilities for existence of each lane. For the lanes with existence probabilities greater than a threshold (0.5 here), the probability map is searched by an interval of 10 rows for the position with the highest probability response to determine accuracy. These positions are then connected by cubic splines, which are the final predictions

4.3. Proposed modification

It is noted in Pan et.al [2] that computational efficiency of SCNN allows it to be easily incorporated into any part of CNN other than its outputs. Also, they note that top hidden layers usually contain information that is both of rich and high semantic value and is an ideal place to apply SCNN.
This forms an architecture where they have used a backbone network before applying the SCNN architectue. In the paper they suggested using VGG16 [13] trained on ImageNet as the backbone of the SCNN network. However, as we know VGG16 is a very deep network with many parameters, which may lead to computational complexity. Thus, our idea in this project was to replace this VGG16 backbone with various models trained on ImageN Accuracy and Loss curves with different backbones

Table 1. Our modification to the SCNN backbone

SCNN backbone	SCNN Size (Mbs)	# layers (OB)	# layers (SB)
VGG16 (Baseline)	157	**44**	44
ResNet18	51	68	**36**
GoogleNet	52	196	54
ShuffleNet	46	167	47
MobileNet	**45**	156	56
ResNext	58	174	81
WideResNet	78	174	81

et [13] and evaluate the performance of the modified architecture on lane detection. Moreover, we did not plan to use the entire pretrained models as backbones. As suggested in [2], the top hidden layer contain rich spatial information. Thus, we sliced the pretrained models to almost a quarter of its depth and attached to the SCNN network in a bid to reduce the complexity of the SCNN network. Then, we ran the models on our dataset and evaluated their performance on our dataset. You can see the model backbone and their characteristics we used in our project in Table 1.

Key: OB: Original Backbone, SB: Sliced Backbone

4.4. Loss Metrics

We know that there are many more non-lane pixels in each image and merely checking the number of pixel predictions would not be a really good metric. Thus, we check two kinds of losses: segmentation loss (for identifying correct pixel position for lanes) and existence loss (for identifying correctly how many lanes exist). For the segmentation loss, we use Cross-Entropy Loss and for existence loss we use Binary version of the Cross-Entropy loss. The general form of Cross-Entropy loss is given by

$$J(w) = \frac{1}{N}\sum_{i=1}^{N} -\left(y_i \log \hat{y}_i + (1-y_i)\log(1-\hat{y}_i)\right) \quad (2)$$

where y_i is the ground truth vector, $\hat{y}_i$ is the predicted vector and N is the total number of training examples.

4.5. Evaluation Metrics

4.5.1 TuSimple Accuracy Metric

We calculated the accuracy of lane prediction using TuSimple's evaluation metric. We first visualize the prediction matrix on images to show predicted lanes and also convert these predictions back JSON labels as required by TuSimple evaluation metric. We obtain the predictions as a matrix and then resize them into the original image size while denoting points with lanes where we exceed a probability threshold. The probability maps are interpolated and searched for interval of 20 rows for the position with the highest probability response to determine accuracy.

$$accuracy = \frac{\sum_{points} correct_{points}}{\#points} \quad (3)$$

Table 2. Hyperparameters for modified SCNN model

Hyperparameter	Value/Type used
Learning Rate	0.015 with learning rate scheduling
Momentum	0.9 (Nesterov)
Weight decay	0.0001
Update Rule	SGD
Batch Size	32

where $points$ refers to points considered for the lane, $correct_{points}$ refer to the prediction values, which are within a certain threshold of the ground truth and $\#points$ is the total number of points being considered for the lanes.

4.5.2 F1-Score

In classification, F1-score is a measure of the test accuracy. It uses the harmonic mean of precision and recall to compute the score. Precision is the number of correct instances from the number of retrieved instances while recall is the total amount of correct instances that were retrieved. The equation is given by

$$F_1 = 2\frac{precision * recall}{precision + recall} \tag{4}$$

where $precision = \frac{TP}{TP+FP}$ and $recall = \frac{TP}{TP+FN}$. Hence, we get

$$F_1 = \frac{2TP}{2TP + FP + FN} \tag{5}$$

where TP is true positive, FP is false positive and FN is false negative. True positives and true negatives are computed based on an accuracy threshold of 0.8.

5. Experiments/Results/Discussion

5.1. Hyperparameters

We evaluated the modified SCNN model on Tusimple Dataset using the hyperparameters tabulated in Table 2. The choice of our hyperparameters was mostly driven by the suggestions in [2]. We tested on a mini-batch size of 32 because a batch size of 64 would often lead to CUDA out of memory errors with a deeper model like SCNN. For the learning rate, we implemented learning rate scheduling with a step scheduling to anneal the learning rate and we chose a smaller weight decay to perform systematic updates. We used SGD with Nesterov Momentum as our optimization algorithm as a standard and as used by [2] Pan et. al.

5.2. Model Evaluation

We ran our model on our dataset for various backbones and realized the training and validation accuracy curves across various epochs. As you can see in Figure 6, validation accuracies roughly track the training accuracies and reached 85% as per the TuSimple Accuracy Metric. It is notable that the baseline model struggles to gain initial validation accuracy, with a visible kink near the 3rd epoch. If you look closer, the corresponding training accuracy also experiences a downfall. This decline in training accuracy might have been amplified in the validation accuracy as weight updates corresponding to the decline in training may have caused the validation losses to increase.
In the last two plots of Figure 6, we have visualized the training and validation loss across all the backbone architectures used. We can notice that the training loss steadily reduces across all iterations. However, validation loss showed high losses for some models like ResNet showing poor performance compared to other models. This is reflected in the lower F1-score for ResNet. Also, notable in the training loss curve are kinks in the loss at specific interval. We noticed higher training losses for the last training batch. Upon inspection, we realized that the reason for this discrepancy was that the last batch of training images was composed of challenging images with low light and occlusions, which were difficult for the network to analyze. In a bid to add some regularization, we had also increased dropout from 0.1 to 0.5 to reduce overfitting.

Table 3. TuSimple accuracy and F1-score

Backbone Used	TuSimple accuracy (%)	F1-score
VGG16(Baseline)	86.52	74.19
ResNet18	85.67	70.53
GoogleNet	84.49	74.82
ShuffleNet	86.75	74.30
MobileNet	85.20	73.50
ResNext	**86.92**	**78.35**
WideResNet	86.39	74.99

5.3. Accuracy Results

5.3.1 TuSimpleAccuracy

As seen in Table 3, we can see that the TuSimple Accuracy metric shows similar accuracies for all backbones including the baseline. This suggests that our hypothesis of choosing parts of pretrained network has worked well in detecting lanes. Moreover, this reaffirms Pan et. al's [2] claim that top hidden layers usually contain information that is both of rich and high semantic value and is an ideal place to apply SCNN. Also, you can note that the model is pretty good at detecting lanes as seen by lane prediction in the rightmost image of Figure 3. However, false positives and false negatives are very important to recognize as the task of lane detection needs to be robust for autonomous driving applications.

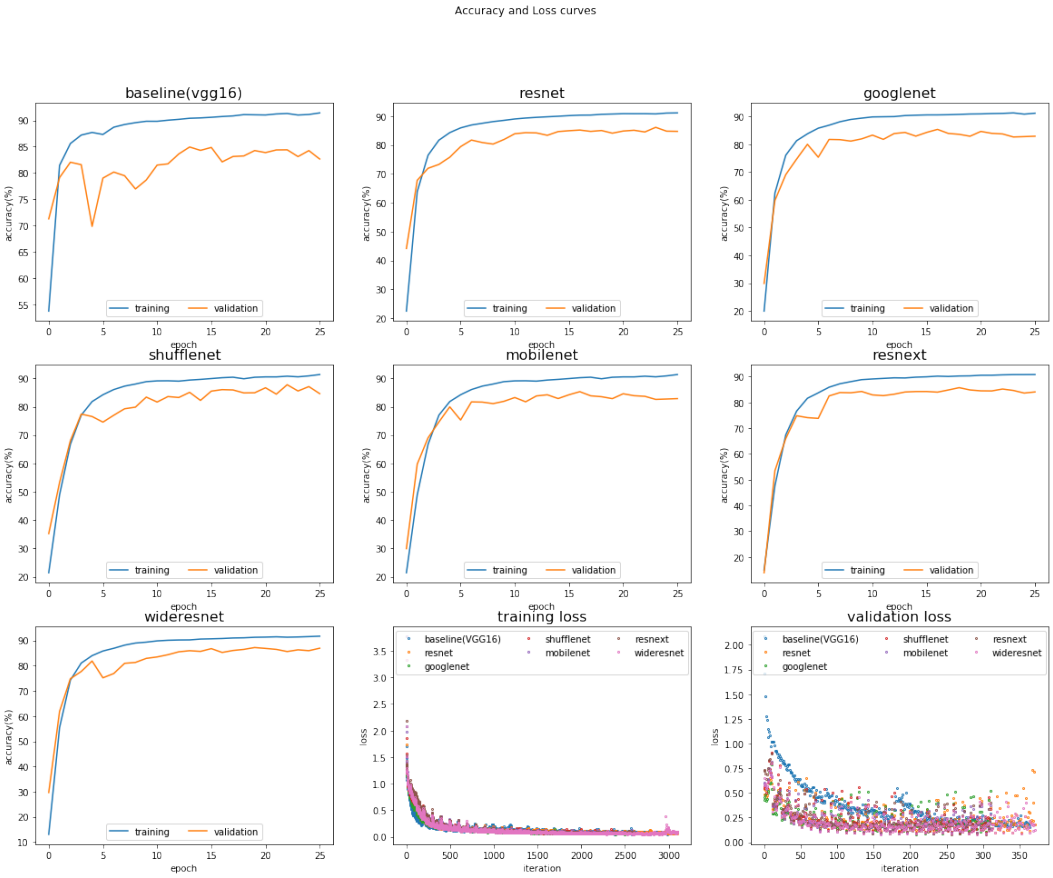

Figure 6. Accuracy and Loss curves with different backbones

5.3.2 F1-score

As discussed in 4.5.2, in addition to Tusimple Accuracy metric, we have also employed F1-score to determine the performance of our model. It is worth noting that similar to TuSimple accuracy, the F1-scores for all the models are also very similar. However, it is interesting to note that TuSimple accuracy and F1-scores are not directly proportional. The highest accuracy and F1-scores are obtained by the ResNext model, with a particularly high F1-score. However, we can safely say that similar F1-scores suggest that we might use any of the tested model backbone's as long as we attach it to the SCNN model for a robust lane prediction.

5.4. Qualitative result visualization

We have included images of some of the predictions of lane markings that we performed on our test set to gain a measure of qualitative performance of our network in Figure 7. As you can see the network performs pretty good in image (a), (b) and (e) even in cloudy conditions. In image (c), the network gets confused due to bright lighting (poor quality image), but recovers in the next frame in image (d). Also, in image (f), the occlusion due to the truck reduces the performance of detection of yellow lane but the network can still identify the lanes correctly.

6. Conclusion/Future Work

In this project, we selected a state-of-the-art SCNN lane detection model developed by Pan et. al [2] and modified its backbone to reduce the model complexity. We explored different model architectures pretrained on ImageNet available in Pytorch [13] and sliced those models to use as the backbone of SCNN to make it less deeper and more simpler. We trained the combined model for several backbones on the TuSimple dataset [1] and observed similar accuracies for both Tusimple Accuracy metric and F1-score. The results shown in Figure 6 and Table 3 confirm our hypothesis that shallow layers usually contain information that is both of rich and high semantic value and is an ideal place to apply SCNN. Thus, by slicing pretrained vision models and only extracting about half of the layers, we were able to match and somewhat exceed the performance of the baseline model. Moreover, we also reduced the model complex-

ity and size as evident in Table 1 while maintaining or exceeding accuracies.
In the future, we may train our model on a more complex and large dataset, which might include complex whether scenarios or different roads from urban and rural landscapes (unlike the TuSimple dataset, which only contains scenes from freeways). Moreover, we may try making modifications to the message passing architecture of the SCNN itself by employing RNNs or LSTMs in a bid to shorten the network. We can also try larger batch sizes on more powerful GPUs to explore the loss behavior with higher batch sizes.
In addition to traditional lane detection approaches, future work might focus on identifying lanes in challenging weather conditions like snow. With the availability of complex datasets growing everyday, relevant datasets may be used to solve the next challenging problems in the lane detection domain.

7. Contributions and Acknowledgements

Aadil worked on dataset research, data pre-processing and data visualization. Ashar worked on data accuracy metric generation and loss and accuracy visualization. Aadil worked on slicing the pretrained models to make them compatible with SCNN and Ashar wrote the script for model training pipeline. Both of us worked equally on proposal of model architecture, model training, final report and final video.
We are relying heavily on the Pytorch implementation of SCNN paper, which can be found here: https://github.com/harryhan618/SCNN_Pytorch
Morever, the evaluation code for Tusimple metric is available here: https://github.com/TuSimple/tusimple-benchmark

We would like to thank the instructors of CS231N, the Teaching Assistants and all the students taking this course, who were very helpful in clearing our doubts and providing valuable suggestions.

References

[1] Tusimple challenge, 2018. https:benchmark.tusimple.ai/.

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

[3] Jihu Kim and Minho Lee. Robust lane detection based on convolutional neural network and random sample consensus. *Neural Information Processing*, pages 454–461, 2014.

[4] J. Li, X. Mei, D. Prokhorov, and D. Tao. Deep neural network for structural prediction and lane detection in traffic scene. *IEEE Transactions on Neural Networks and Learning Systems*, 28(3):690–703, 2017.

[5] Alexandru Gurghian, Tejaswi Koduri, Smita V. Bailur, Kyle J. Carey, and Vidya N. Murali. Deeplanes: End-to-end lane position estimation using deep neural networksa. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2016.

[6] Jinju Zang, Wei Zhou, Guanwen Zhang, and Zhemin Duan. Traffic lane detection using fully convolutional neural network. pages 305–311, 11 2018.

[7] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(12):2481–2495, 2017.

[8] Fabio Pizzati, Marco Allodi, Alejandro Barrera, and Fernando García. Lane detection and classification using cascaded cnns. *arXiv preprint arXiv:1907.01294*, 2019.

[9] D. Neven, B. D. Brabandere, S. Georgoulis, M. Proesmans, and L. V. Gool. Towards end-to-end lane detection: an instance segmentation approach. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 286–291, 2018.

[10] Yuenan Hou, Zheng Ma, Chunxiao Liu, and Chen Change Loy. Learning lightweight lane detection cnns by self attention distillation. In *The IEEE International Conference on Computer Vision (ICCV)*, October 2019.

[11] Noa Garnett, Rafi Cohen, Tomer Pe'er, Roee Lahav, and Dan Levi. 3d-lanenet: End-to-end 3d multiple lane detection. In *The IEEE International Conference on Computer Vision (ICCV)*, October 2019.

[12] Qin Zou, Hanwen Jiang, Qiyu Dai, Yuanhao Yue, Long Chen, and Qian Wang. Robust lane detection from continuous driving scenes using deep neural networks. *IEEE Transactions on Vehicular Technology*, 2019.

[13] PyTorch. torchvision.models, 2020.

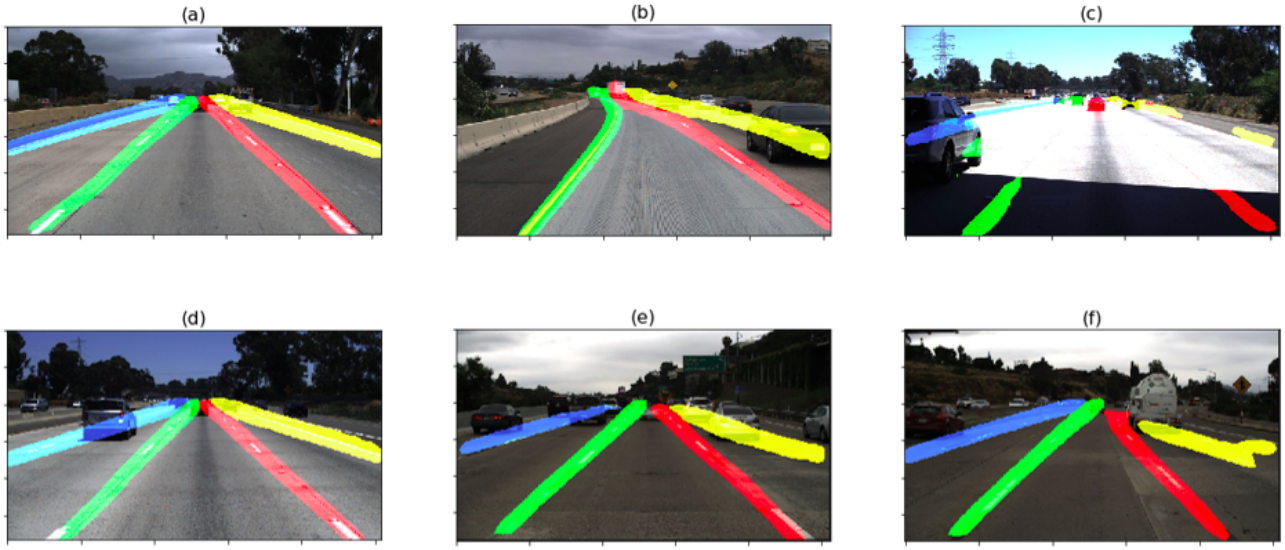

Figure 7. Lanes predicted by our model on various images