End to End Video Segmentation for Driving : Lane Detection For Autonomous Car

Tejas Mahale Chaoran Chen and Wenhui Zhang * equal contribution
Penn State University
State College, Pennsylvania
wenhui@gwmail.gwu.edu

ABSTRACT

Safety and decline of road traffic accidents remain important issues of autonomous driving. Statistics show that unintended lane departure is a leading cause of worldwide motor vehicle collisions, making lane detection the most promising and challenge task for self-driving. Today, numerous groups are combining deep learning techniques with computer vision problems to solve self-driving problems. In this paper, a Global Convolution Networks (GCN) model is used to address both classification and localization issues for semantic segmentation of lane. We are using color-based segmentation is presented and the usability of the model is evaluated. A residual-based boundary refinement and Adam optimization is also used to achieve state-of-art performance. As normal cars could not afford GPUs on the car, and training session for a particular road could be shared by several cars. We propose a framework to get it work in real world. We build a real time video transfer system to get video from the car, get the model trained in edge server (which is equipped with GPUs), and send the trained model back to the car.

KEYWORDS

Autonomous car, Lane detection, Color-based segmentation

ACM Reference Format:
Tejas Mahale Chaoran Chen and Wenhui Zhang * equal contribution. 2018. End to End Video Segmentation for Driving : Lane Detection For Autonomous Car. In *Proceedings of Penn State Deep Learning Fall 2018 (Deep Learning' 18).* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Statistics show that unintended lane departure is a leading cause of worldwide vehicle collisions, which leads to substantial financial costs to both society and the individuals involved [1]. To reduce the number of traffic accidents and to improve safety, research on Driver Assistance System (DAS) have been conducted worldwide for many years and even expanded to autonomous cars. Autonomous car is always equipped with intelligent vehicle (IV) system, and lane detection is an important information for this system. In this system, some tasks including road following, keeping within the correct lane, maintaining a safe distance between vehicles, controlling the speed of a vehicle according to traffic conditions and road conditions, moving across lanes in order to overtake vehicles and avoid obstacles, searching for the correct and shortest route to a destination have to be completed [2]. The premise of all the tasks are high related to the lanes. Only when we know the accurate lanes will we finish above tasks better. Furthermore, The distance between the lane boundaries and obstacle is also important because it makes sure that the vehicle is in a safe distance from another vehicle or obstacles to avoid any possible collisions[2]. To obtain the information of the distance to the lane boundaries, we need to know the lane boundaries first. That is why lane detection is an important part of intelligent vehicle system.

Lane detection mainly focus on locating both the center line and the edge of each lane in a road image. lane markings help differentiate the lanes from other characteristic objects on the road such as other vehicles, pedestrian, and animals running into the road. But there are some challenges we have to face. Firstly, does markings really mark the correct lane? As we know, there are various road conditions and environment conditions that lanes maybe not visible in a image. Sometimes the quality of the image can also be the reason. Apart from that, road splitting or merging and the interference from roadside objects or shadows could be another problem which will worsen the detection.

However, the great variety of road environments necessitates the use of complex vision algorithms that not only requires expensive hardware to implement but also relies on many adjustable parameters that are typically determined from experience [3]. With the development of the deep learning techniques, numerous groups have applied a variety of deep learning techniques to computer vision problems [4].

In this paper, we gather original data from Carla simulator In Implementation Details section, we do some pre-processing work to clean the data, which is a preparation for using a image segmentation techniques to deal with the lane detection problem. The method is based on color to differentiate different objects. Then in Experiments and Evaluation section, we conduct a Global Convolution Networks (GCN) model using color-based segmentation and use Adam as an optimizer to optimize the results. After obtaining the results(using test data to get training image), we compare the image we get with the actual image to evaluate the model.

In summary, we highlight our potential contributions below:

- This report introduces a fine grained architecture of GCN model;
- We implements a GCN network with optimization method of Residual-based boundary refinement and Adam optimization;

Deep Learning' 18, Fall 2018, State College, PA, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

- This paper discusses and conducts performance diagnostics of lane detection tasks under various weather conditions.
- We proposed a framework to get the model trained and tested in real world.

The rest of this report is structured as follows. Section 2 gives out overview for related works. Section 3 explains and gives out an overview on algorithms and math background of our model. In Section 4 , we present our detailed implementation for data pre-processing pipeline and data augmentation. In Section 5 , we present our proposal for evaluating performance analysis for our lane detection model. It also explains performance of our model on train, validation and test data.

2 RELATED WORK

2.1 Image Segmentation Techniques

Image segmentation is the operation of partitioning an image into a collection of connected sets of pixels. Main methods of region segmentation includes, region growing, clustering and split and merge. Region growing techniques start with one pixel of a potential region and try to grow it by adding adjacent pixels till the pixels being compared are too far from similar. The first pixel selected can be just the first unlabeled pixel in the image or a set of seed pixels can be chosen from the image. Usually a statistical test is used to decide which pixels can be added to a region. [6] Clustering methods includes K-means Clustering and Variants, Isodata Clustering, Histogram-Based Clustering and Recursive Variant and Global-Theoretic Clustering. [7] In some image sets, lines, curves, and circular arcs are more useful than regions or helpful in addition to regions. Thus people use lines and arcs segmentation as basis of image segmentation. Basic idea is looking for a neighborhood with strong signs of change, and detect them as edge. Edge detectors are based on differential operators. Differential operators attempt to approximate the gradient at a pixel via masks. And then threshold the gradient to select the edge pixels. One widely used edge detector used in auto driving lane detection in earlier days is Canny Edge Detector (CED). CED firstly smooth the image with a Gaussian filter, then compute gradient magnitude and direction at each pixel of the smoothed image. It zeros out any pixel response smaller than the two neighboring pixels on either side of it, along the direction of the gradient, and track high-magnitude contours. Afterwards it keeps only pixels along these contours, so weak little segments go away. [8]

These classical image segmentation methods are good, however they do not work well with raining or snowing road situations. It is hard for these classical image segmentation methods to handle various scenarios. Thus higher order image segmentation methods, such as deep learning's convolution neural network (CNN) is a must in auto driving.

2.2 Structured Deep Learning

Classical machine learning on structured datasets methods includes kernel-based methods and graph-based regularization techniques [1, 2]. However these machine learning requires a lot of feature engineering before certain tasks such as image classification. Furthermore, most of the time these features require domain knowledge, creativity and a lot of trial and error. Structured Deep Learning (SDL) is a fast, no domain knowledge requiring, and high performing machine learning method.

In the last couple of years, a number of papers re-visited machine learning on structured datasets, like graphs. Some works has done in generalizing neural networks to work on arbitrarily structured graphs [4, 5]. Some of them achieve promising results in domains that have previously been dominated by classical machine learning methods mentioned above. Kipf [9] came up with a graph convolutions method, which is good for image segmentation. Global convolutions are generalization of convolutions, and easiest to define in spectral domain. General Fourier transform scales poorly with size of data so we need relaxations. In this paper they use first order approximation in Fourier-domain to obtain an efficient linear-time graph-CNNs. We adapt this approach in our paper. We illustrate here what this first-order approximation amounts to on a 2D lattice one would normally use for image processing, where actual spatial convolutions are easy to compute in this application the modelling power of the proposed graph conv-networks is severely impoverished, due to the first-order and other approximations made. It uses concept of a convolution filter for image pixels or a linear array of signals.

2.3 Optimization Techniques on Gradients

Optimization algorithms used to accelerate convergence includes stochastic gradient methods and stochastic momentum methods. Stochastic gradient methods can generally be written

$$w_{k+1} = w_k - \alpha_k \tilde{\nabla} f(w_k), \tag{1}$$

where $\tilde{\nabla} f(w_k) := \nabla f(w_k; x_{i_k})$ is the gradient of some loss function f computed on a batch of data x_{i_k}.

Stochastic momentum methods have been used to accelerate training. These methods could be written as

$$w_{k+1} = w_k - \alpha_k \tilde{\nabla} f(w_k + \gamma_k(w_k - w_{k-1})) + \beta_k(w_k - w_{k-1}). \tag{2}$$

Sequence of iterates (2) includes Polyak's heavy-ball method (HB) with $\gamma_k = 0$, and Nesterov's Accelerated Gradient method (NAG) with $\gamma_k = \beta_k$.

However there are some exceptions as well, such as (1) and (2). These are adaptive gradient and adaptive momentum methods. These methods construct the entire sequence of iterates, say $(w_1, \cdots, w_k)$ as a local distance measure. AdaGrad [10], RMSProp [11], and Adam [12] can generally be written as

$$w_{k+1} = w_k - \alpha_k \mathrm{H}_k^{-1} \tilde{\nabla} f' + \beta_k \mathrm{H}_k^{-1} \mathrm{H}_{k-1}(w_k - w_{k-1}), \tag{3}$$

where $\tilde{\nabla} f'$ is $\tilde{\nabla} f(w_k + \gamma_k(w_k - w_{k-1}))$, and $\mathrm{H}_k := H(w_1, \cdots, w_k)$ is a positive definite matrix. H_k is a diagonal matrix. Its entries are defined as square roots of a linear combination of squares of past gradient components.

In deep learning acceleration of convergence, specific settings of the parameters are stated in Table 1. In this table, $D_k = (g_k \circ g_k)$ and $G_k = H_k \circ H_k$.

As we could see from Table 1, adaptive methods change and adapt to geometry of the data. However, stochastic gradient descent and related variants use the ℓ_2 geometry inherent. This is considered as equivalent to making$\mathrm{H}_k = \mathrm{I}$ in adaptive methods. Performance is

defined as loss function than the function f used in training. Thus, it seems like Adam is more favorable in our GCN project.

	SGD	HB	NAG	AdaGrad	RMSProp	Adam
G_k	I	I	I	$G_{k-1} + D_k$	$\beta_2 G_{k-1} + (1-\beta_2) D_k$	$\frac{\beta_2}{1-\beta_2^k} G_{k-1} + \frac{(1-\beta_2)}{1-\beta_2^k} D_k$
α_k	α	α	α	α	α	$\alpha \frac{1-\beta_1}{1-\beta_1^k}$
β_k	0	β	β	0	0	$\frac{\beta_1(1-\beta_1^{k-1})}{1-\beta_1^k}$
γ	0	0	β	0	0	0

Table 1: Parameter Settings of Optimization Techniques used in Deep Learning.

3 FORMULATION

Lane detection involves following steps. Firstly, compute the camera calibration matrix and distortion coefficients given a set of chessboard images. Then, apply a distortion correction to raw images, and use color transforms, gradients, etc., and apply deep learning for classification analysis to create a threshold binary image. Afterwards, from a birds-eye view, apply a perspective transform to rectify binary image. Then, detect lane pixels and fit to find the lane boundary, and determine the curvature of the lane and vehicle position with respect to center. The last step is to warp the detected lane boundaries back onto the original image. Last but not least, put visual display of the lane boundaries and numerical estimation of lane curvature and vehicle position.

In this paper, the GCN utilizes connectivity structure of graph as the filter to perform neighborhood mixing. Its architecture may be elegantly summarized as:

$$H^{(l+1)} = \sigma(\hat{A}H^{(l)}W^{(l)}),$$

where $\hat{A}$ is normalization of the graph adjacency matrix, $H^{(l)}$ is row-wise embedding of graph vertices in the lth layer, $W^{(l)}$ is a parameter matrix, and σ is non-linearity.

in this paper we consider a building block defined as: $y = F(x, W_i) + x$. Here x and y are the input and output vectors of a particular layer. The function $F(x, W_i)$ represents learning residual mapping. $F = W_2 * \sigma(W_1 x)$ in which σ denotes ReLU and the biases are omitted for simplifying notations. The operation $F + x$ is performed by a shortcut connection and element-wise addition. We adopt the second non-linearity after the addition.

In this section, we first propose a novel GCN to address the contradictory aspects — classification and localization in semantic segmentation. Then using GCN 1 we design a fully-convolutional framework for semantic segmentation task. Comparing with other many semantic segmentation models, our model has faster training speeds with smaller model size, while keeping similar accuracy. In features extraction stage, we use the max pooling module following by conv-layers and an up-sampling layer. After these two modules, feature maps has same size as input image. When the enlargement factor is large, it brings noise and make boundary pixels of two areas difficult to classify. And that is why we add refinement module at the end of the model to refine the segmentation area 2 In refinement module, we use combination of convolution layers and

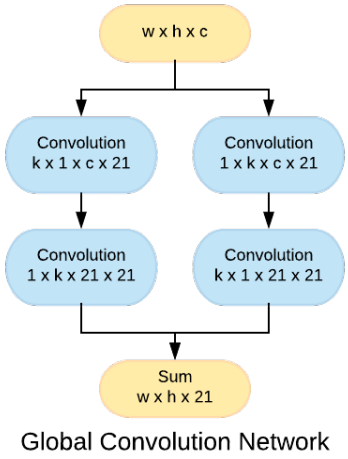

Figure 1: Details of GCN

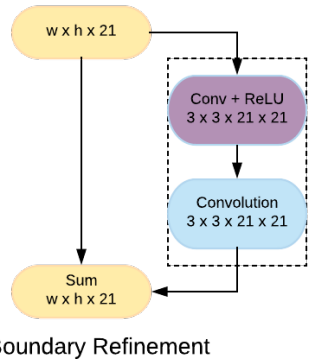

Figure 2: Details of BR

pooling layers, with a conv-net with ReLU, and then process it with convolution of size 3x3x21x21.

In convolution neural network training, network weights are adjusted by loss function evaluation. As images vary from class to class, influence of each class to the loss is different. In each iteration, we calculate the weights based on current input batch. Weights are different in each iteration. The weights is calculated as in Eq.4.

$$w_i = \begin{cases} 1, & n_i = 0 \\ \beta, & w_i < \beta \\ \dfrac{N}{2 * c * n_i}, & \beta < w_i < \alpha \\ \alpha, & w_i > \alpha \end{cases} \tag{4}$$

Where, w_i is weight of class i, c is class number, and value of i is from 0 to c. β and α are lower and upper threshold of w_i, we set threshold to avoid excessive weights differences. N is the total pixel number of this batch, n_i is the pixel number of class i, when $n_i = 0$, it means that the class i does not appear in this

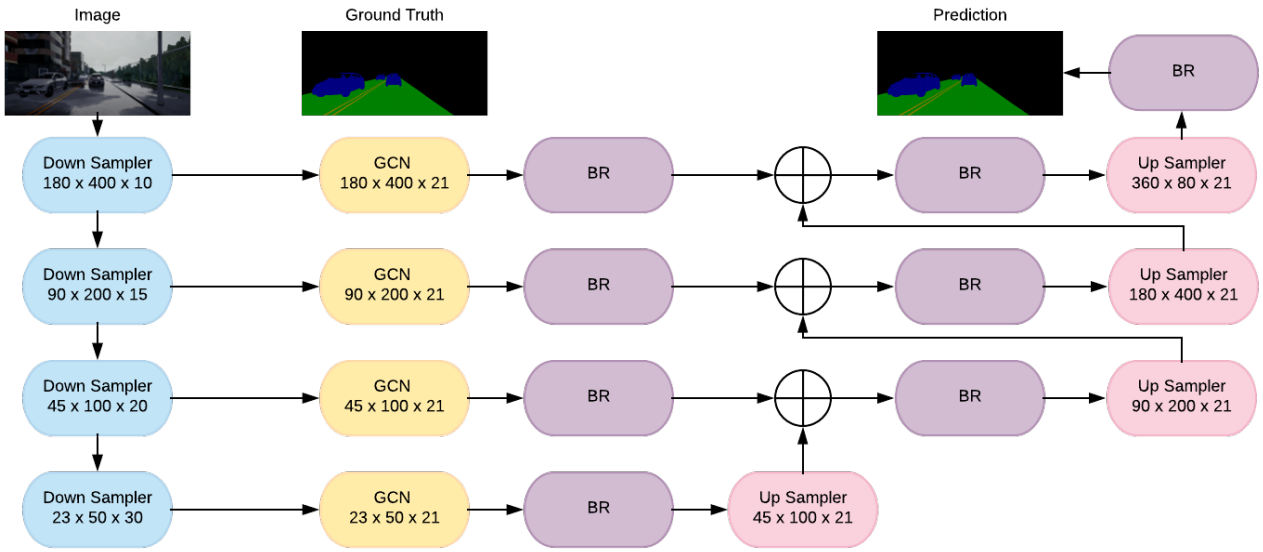

Figure 3: An overview of the whole pipeline.

batch, we set the weight to 1. Because we need to increase the effect of small pixel number class on loss, so the smaller the n_i, the larger the w_i is. N and c are constant, w_i is just changed by n_i. When the n_i is the average number, w_i is calculated to be $\frac{1}{2}$, the multiplicative coefficient of $\frac{1}{2}$ is also used to decrease the w_i of large pixel number of class. The loss function is shown in Eq.5, where x_{ij}, y_{ij} are prediction class and label in pixel (i, j), w is the loss weights.

$$LOSS = \sum_{i=1} \sum_{j=1} w \left\| x_{ij} - y_{ij} \right\|^2 \tag{5}$$

And the whole architecture could be seen as shown in Fig 3.

3.1 Encode - Decoder

To decodify our depth first we get the int24. $R+G*256+B*256*256$. Then normalize it in the range $[0, 1]$. $Ans/(256*256*256-1)$. And finally multiply for the units that we want to get. We have set the far plane at 1000 metres. Ans * far. Encoded stage includes encoding image to segmented image. Every pixel of label image is red channel class label. We use retained class labels with road and vehicle and converted them to green and blue respectively. Encoder network consist of convolution layers, batch normalization and max pooling for down-sampling[13]. For classification problems, multiple convolution layers with decrease in size of kernel and increase in number of filters with each layer pooling is advised[14]. On other hand decoder block consists of strided convolution and up-sampler which will re-size image into original shape. When we transfer the trained image back to color map to be project to videos, we just use $blue = value/(256*256)$, $green = (value - blue*256*256)/256$ and $red = value - green*256 + blue*256*256$.

3.2 GCN and BR

There are two tasks in segmentation, classification and localization. Conical CNN architectures are good for classifications. In case of object segmentation, large kernel size plays important role in term of object localization[15]. But usage of large kernels is heavy on weights of model as kernel size increases, number of parameters also increases. To achieve global convolution (GCN) performance with less weights, GCN was divided into two one dimensional kernels. Effect of both kernels added at end to get actual GCN output. Boundary refinement block(BR) is residual block which helps to get boundary structure of shape properly.

3.3 Adam

In this paper, we use Adam. Similar to momentum, it keeps exponentially decaying of past gradients m_t. It compute bias-corrected first and second moment estimates:

$$\begin{aligned} \hat{m}_t &= \frac{m_t}{1-\beta_1^t} \\ \hat{v}_t &= \frac{v_t}{1-\beta_2^t} \end{aligned} \tag{6}$$

m_t and v_t are estimates of the first moment and the second moment of the gradients respectively. Parameters are updated using:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t \tag{7}$$

In this paper, we use default values of 0.9 for β_1, 0.999 for β_2, and 10^{-8} for ϵ.

4 IMPLEMENTATION DETAILS

4.1 Dataset

First, we gathered our 3000 road images from a platform named Carla Simulator. It is an open-source simulator especially for autonomous driving research. It is a good tool since it provides various different images in a variety of circumstances including different types of weather conditions and different time of day. Through driving a car in urban conditions, we could get train images(shown in Figure 4) and corresponding encoded label images(shown in Figure 5). 3,000 images along with semantic labels with resolution 600 x 800 x 3 are collected as our dataset.

Figure 4: original image

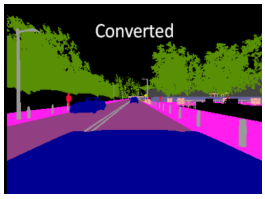

Figure 5: converted image

Value	Tag
0	None
1	Buildings
2	Fences
3	Other
4	Pedestrians
5	Poles
6	RoadLines
7	Roads
8	Sidewalks
9	Vegetation
10	Vehicles
11	Walls
12	TrafficSigns

Figure 8: Tags

4.2 Data Pre-processing

To reduce the workload of computer and accelerate the running speed, car hood and sky portion are cropped in each image for 240*800 pixels(From Figure 6 to Figure 7). That leads to a dataset consisting of 360*800 images of the red, green, and blue color channels.

Figure 6: 600*800*3

Figure 7: 360*800*3

The server provides an image with the tag information encoded in the red channel. A pixel with a red value of x displays an object with tag x. The following tags are displayed (shown in Figure 8). For training purposes, encoded image(shown in Figure 9) should be transformed to segmented image(shown in Figure 10). We retained class labels with road and vehicle and converted them to green and blue respectively.

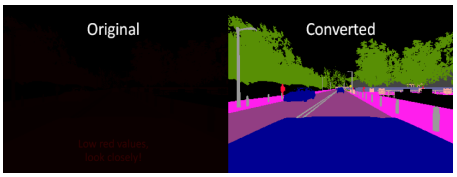

Figure 9: encoded image to Segmented image(1)

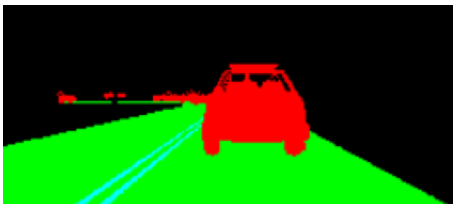

Figure 10: encoded image to Segmented image(2)

4.3 Data Augmentation

To ensure the diversity of dataset, we made some data augmentation. As is shown in Figure 11 and Figure 13, Rotation (0, 30 degree) like that is given to the original dataset to get another 3000 images, while in Figure 12 and Figure 14, shifting (Width = (0, 0.2), height = (0, 0.1)) like that is given to the original dataset as well to get other 3000 images. In total, we obtained 9000 images dataset at present, which is of course a large amount of images and also make sure the diversity of the dataset.

Figure 11: Rotation

Figure 12: Shifting

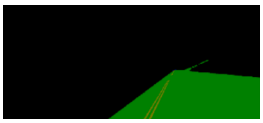

Figure 13: Rotation

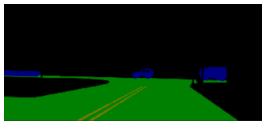

Figure 14: shifting

5 EVALUATION AND DISCUSSION

5.1 Metric

In this paper, evaluation on both Global Convolution Network and Boundary refinement are conducted. We evaluate our GCN based lane detection model in terms of minimum loss performance with help of minimum square error and mean average error. Our experiment setup is composed of one class of server. For training and experiments performed on Tesla P100-PCIE gpu memoryclock rate 1.3285 with 11.95 GB video memory on Linux distribution.Conda environment is used with tensorflow gpu v1.10, Keras gpu v2.2.2, python version 3.5.

Our implementation for training is as follows. We re-size images with its shorter side randomly sampled in 600 x 800 x 3 for scale augmentation 360 x 800 x 3 resolution. The standard color augmentation mentioned in previous section is used. Batch normalization is adopted right after each convolution layer before activation. We initialize the weights as per Xavier initializer, and train encoder nets from scratch. We use Adam with a global-batch size of 64. The learning rate starts from 0.001 and the models are trained for up to 40 iterations. We use a weight decay of 0.9 and a momentum of 0.999.

In order to judge whether a lane segmentation is successfully detected, we view lane markings as two classified value and calculate the Minimum Square Error (MSE) and Mean Absolute Error (MAE) between the ground truth and the prediction. MSE is average of square of pixel to pixel difference between two images where as MAE is pixel to pixel absolute difference between images.

$$\text{MSE} = \frac{1}{N}\sum_{n=1}^{N}(Y_{true} - Y_{predicted})^2$$

$$\text{MAE} = \frac{1}{N}\sum_{n=1}^{N}|(Y_{true} - Y_{predicted})|$$

5.2 Hyper-parameter tuning

Hyper-parameter tuning is important process of deep learning algorithms. In this project, we have implemented encoder-decoder architecture which has parameters like number of filters in each encoder layer, filter size, learning rate of optimizer, number of iterations to get minimum loss.

Parameter	Value
Optimizer	Adam
Learning rate	0.001
Number of iterations	40
Number of Filters per layer	8, 16, 20, 32

Table 5.1

As explained earlier, we implemented conical structure for encoder blocks which consist of convolution layer, batch normalization and max pooling to half the image resolution at every step along with gradual increase in number of filters. We trained our model for 5-10 iterations of various combination of hyper-parameter and observed the decay in mean square loss. Finally we came with hyper-parameter settings that give minimum loss at 40th iteration.

5.3 Training and Validation Results

We divided our image dataset into train, validation and test categories. We trained our model on train data and tuned hyper-parameter on validation data.

Minimum Square Error (MSE)

Iterations	Training (MSE)	Validation (MSE)
0 Initial loss	1592.1951	1443.7032
10 Iterations	192.1637	337.5560
20 Iterations	102.3213	119.4761
30 Iterations	74.0532	89.0769
40 Iterations	42.7771	61.4360

Table 5.2

Mean Average Error (MAE)

Iterations	Training (MAE)	Validation (MAE)
0 Initial loss	13.3800	13.3908
10 Iterations	3.1369	6.8221
20 Iterations	2.9696	3.7389
30 Iterations	2.0202	2.8999
40 Iterations	1.7011	2.4999

Table 5.3

After 40 iterations, train loss was further decreasing but on same time validation loss started increasing, so we had to stop training process. We can see final iteration values of MAE and MSE and conclude that we have got optimized parameters with minimum loss and less over-fitting.

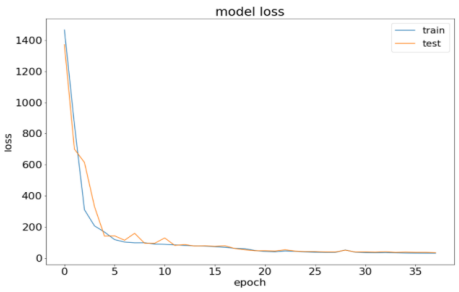

Figure 15: MSE for Train and Validation

5.4 Test Results

Once we concurred best hyper-parameters for minimum loss on given model with help of MSE and MAE values, we tested our model on reserved test dataset to get unbiased evaluation of our model.

Metric	Test set evaluation
Minimum Square Error	57.5875
Mean Absolute Error	2.2104

Table 5.4

From table 5.2 and 5.4, we can compare MSE values on train, validation and test set. There is not much difference in train and validation MSE. Also, validation and train values are almost in range. It implies that model is not facing over-fitting problem.

Figure 16: Test image 1

Figure 17: Test Image 2

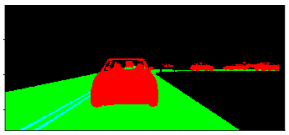

Figure 18: Ground truth of image 1

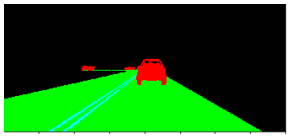

Figure 19: Ground truth of Image 2

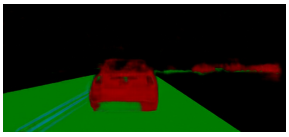

Figure 20: Model Prediction image 1
MSE = 48.8590

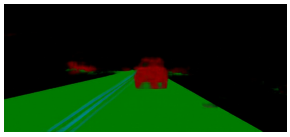

Figure 21: Model Prediction image 2
MSE = 63.3626

Method	MSE
FCN 8s	65.3
DPN	59.1
CRFasRNN	62.5
Scale invariant *CNN + CRF*	66.3
Dilation10	67.1
DeepLabv2-CRF	70.4
Adelaide_context	71.6
LRR-4x	71.8
Large-Batch	76.9
Our approach	**57.5875**

Table 5.5 Experimental results on test set

We calculated minimum square error for test image1 and test image2 from test dataset. Both images were able to capture shape of vehicle as well as lane boundaries. From ground truth label image and model predicted image, we compared MSE score of both images.

6 CONCLUSION

In this paper, a Global Convolution Networks (GCN) model is used to address both classification and localization issues in lane detection. A residual-based boundary refinement and Adam optimization is also used to achieve 57.5875 performance. We also build a picamera on RaspberryPi and put it on a car, this will send real time video to our edge server. On edge server side, we will have training scripts running and sending trained GCN model back to the car. The code could be found here:

https : //github.com/wenhuizhang/auto_driving_car

7 REFERENCES

[1] Narote, S. P., Bhujbal, P. N., Narote, A. S., & Dhane, D. M. (2018). A review of recent advances in lane detection and departure warning system. Pattern Recognition, 73, 216-234.

[2] Chiu, K. Y., Lin, S. F. (2005, June). Lane detection using color-based segmentation. In Intelligent Vehicles Symposium, 2005. Proceedings. IEEE (pp. 706-711).

[3] Yim, Y. U., Oh, S. Y. (2003). Three-feature based automatic lane detection algorithm (TFALDA) for autonomous driving. IEEE Transactions on Intelligent Transportation Systems, 4(4), 219-225.

[4]Huval, B., Wang, T., Tandon, S., Kiske, J., Song, W., Pazhayampallil, J., ... & Mujica, F. (2015). An empirical evaluation of deep learning on highway driving. arXiv preprint arXiv:1504.01716.

[3] Martin Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015.

[4] Bruna, J., Zaremba, W., Szlam, A., & LeCun, Y. (2013). Spectral networks and locally connected networks on graphs. arXiv preprint arXiv:1312.6203.

[5] Henaff, M., Bruna, J., & LeCun, Y. (2015). Deep convolutional networks on graph-structured data. arXiv preprint arXiv:1506.05163.

[6] Al-Hujazi, E., & Sood, A. (1990). Range image segmentation combining edge-detection and region-growing techniques with applications sto robot bin-picking using vacuum gripper. IEEE transactions on systems, man, and cybernetics, 20(6), 1313-1325.

[7] Lucchese, L., & Mitra, S. K. (2001). Colour image segmentation: a state-of-the-art survey. Proceedings-Indian National Science Academy Part A, 67(2), 207-222.

[8] Ding, L., & Goshtasby, A. (2001). On the Canny edge detector. Pattern Recognition, 34(3), 721-725.

[9] Kipf, T. N., & Welling, M. (2016). Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907.

[10] Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. Journal of Machine Learning Research, 12(Jul), 2121-2159.

[11] Tieleman, T., & Hinton, G. (2012). Rmsprop: Divide the gradient by a running average of its recent magnitude. coursera: Neural networks for machine learning. COURSERA Neural Networks Mach. Learn.

[12] Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

[13] âĂIJAutoencoders, Unsupervised Learning, and Deep ArchitecturesâĂİ, Pierre Baldi, 2012

[14] "Very Deep Convolutional Networks for Large-Scale Image Recognition", Karen Simonyan, Andrew Zisserman, September 2014

[15] "Large Kernel Matters – Improve Semantic Segmentation by Global Convolutional Network", Chao Peng, Xiangyu Zhang, Gang Yu, Guiming Luo, Jian Sun, March 17