Road Segmentation Using CNN and Distributed LSTM

Yecheng Lyu, Lin Bai and Xinming Huang
Department of Electrical and Computer Engineering
Worcester Polytechnic Institute
Worcester, MA 01609, USA
{ylyu,lbai2,xhuang}@wpi.edu

***Abstract*—In automated driving systems (ADS) and advanced driver-assistance systems (ADAS), an efficient road segmentation is necessary to perceive the drivable region and build an occupancy map for path planning. The existing algorithms implement gigantic convolutional neural networks (CNNs) that are computationally expensive and time consuming. In this paper, we introduced distributed LSTM, a neural network widely used in audio and video processing, to process rows and columns in images and feature maps. We then propose a new network combining the convolutional and distributed LSTM layers to solve the road segmentation problem. In the end, the network is trained and tested in KITTI road benchmark. The result shows that the combined structure enhances the feature extraction and processing but takes less processing time than pure CNN structure.**

***Index Terms*—Autonomous vehicle, road segmentation, CNN, LSTM**

I. Introduction

In recent years, growing research interest is witnessed in automated driving systems (ADS) and advanced driver-assistance systems (ADAS). As one of the essential modules, road segmentation perceives the surroundings, detects the drivable region and builds an occupancy map [1] [2] [3] [4]. A drivable region is a connected road surface area that is not occupied by any vehicles, pedestrians, cyclists or other obstacles. In the ADS workflow, road segmentation contributes to other perception modules and generates an occupancy map for planning modules. Therefore, an accurate and efficient road segmentation is necessary.

Camera-based road segmentation has been investigated for decades since cameras generate high-resolution frames frequently and they are cost effective. Traditional computer vision algorithms employed manually defined features such as edges [5] and histogram [6] for road segmentation. Those features, however, worked on limited situations and were difficult to extend to new scenarios [1].

Convolutional neural network (CNN) based algorithms attracted research interest in recent years. By implementing massive convolutional kernels to a deep neural network, CNNs are capable to handle various driving scenarios. Existing CNN based road segmentation algorithms such as FCN [7], SegNet [8], StixelNet [9], Up-conv-Poly [10] and MAP [11] generated a precise drivable region but required large computational. Table II presents their performance on KITTI road benchmark as well as their parameter counts, floating-point operations and running time for each frame processing. Recent research proposed several efficient networks and network structures such as MobileNet [12] and Xception [13]. However, they were still too large to work on embedded systems. In our previous work [14] we explored the use of CNN stack to process the camera data and implemented on a VLSI die but experienced high memory usage.

Long-Short Term Memory (LSTM) is a kind of recurrent neural networks (RNNs) that are often used to process streaming data such as an audio signal and video sequences. By introducing memory cells and gates, LSTM units are capable to extract context features in a long sequence of inputs. Recently, distributed LSTMs are introduced to share the LSTM kernel weights in multi-sequence processing. However, LSTM and distributed LSTM algorithms only focus on time-domain processing.

In this paper, we introduce the distributed LSTM to work on spatial domain and process row sequences on camera frames and corresponding feature maps. This is one of the first efforts to LSTM on spatial sequence processing. We also propose a deep neural network that combines the convolutional layer and distributed LSTM layers on the spatial domain to solve road segmentation tasks. The proposed network is trained and tested on the KITTI road benchmark [15]. It is shown that the proposed method achieves comparable accuracy to the existing solutions but takes fewer calculations and less processing time. The result shows that LSTM structures significantly enhance the context feature extraction in a large feature map. The rest of the paper is organized as follows: Section IIcompares the scheme of the convolutional layer and the distributed LSTM layer on feature map processing. Section III introduces the proposed network that combines CNN and distributed LSTM. In Section IV, we evaluate the proposed network in the KITTI road benchmark. Section V concludes the paper.

II. Comparison of convolutional layer and distributed LSTM layer

In this section, we compare the scheme, sensing area and computational complexity of CNN and distributed LSTM structure in a feature map processing and try to show their advantages and disadvantages in feature map processing.

Suppose we have a feature map that is formatted as a $w_1 \times h_1 \times d_1$ tensor. A convolutional kernel of size $w_k \times h_k \times d_x$,

978-1-7281-0397-6/19/$31.00 ©2019 IEEE

stride (s_w, s_h) and padding (p_w, p_h) is designed to process the feature map. The scheme is described in Figure 1(a). The kernel initially convolutes the first patch in red and generates a $1 \times 1 \times d_k$ vector. Then the kernel strides certain pixels to the right and convolutes the next patch. After all available patches are convoluted, a $w_2 \times h_2 \times d_2$ output tensor is generated. (1) - (3) presents the relationship between the size of output tensor, input tensor and kernel settings. In each step there are $w_k \times h_k \times d_1$ inputs contribute to a $1 \times 1 \times d_k$ output vector. Floating-point operations for this layer is $w_2 \times h_2 \times d_1 \times w_k \times h_k \times d_k \times 2$.

$$w_2 = \left[\frac{w_1 + 2p_w}{s_w}\right] \tag{1}$$

$$h_2 = \left[\frac{h_1 + 2p_h}{s_h}\right] \tag{2}$$

$$d_2 = d_k \tag{3}$$

On the other hand, suppose we have a feature map of the same size, and we now have h_1 distributed LSTM units with d_l memory cells. The units share the same weights and bias but work on different rows of the feature map. In addition, we configure the LSTM units to return an output vector in each step. The scheme is illustrated in Figure 1(b). In each step, each distributed LSTM unit inputs a $1 \times 1 \times d_1$ vector, updates its memory states through gate functions and output a $1 \times 1 \times d_l$ vector. Therefore, the distributed LSTM layer generates a $w_1 \times h_1 \times d_l$ output tensor. In the i^{th} step, there are $i \times d_1$ inputs contribute to a $1 \times 1 \times d_l$ output tensor. Floating-point operations for this layer is $w_1 \times h_1 \times 8 \times d_1 \times d_l$.

Comparing the convolutional layer and distributed LSTM layer, we have the following conclusions:

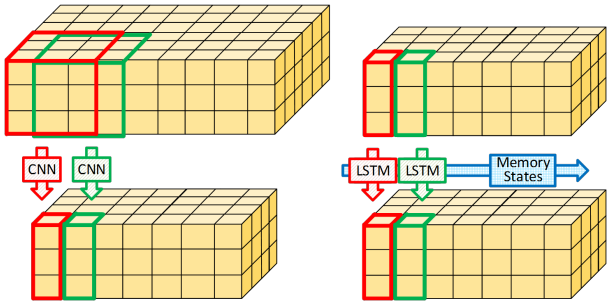
(1) If convolutional layer and distributed LSTM layer have the same output size, the ratio of the floating-point operations is $w_k \times h_k : 4$. Considering that $w_k \geqslant 3$ and $h_k \geqslant 3$, a distributed LSTM layer save at least 56% floating-point operations than a convolutional layer.

(2) If convolutional layer and distribute LSTM layer have the same output size, the ratio of the average number of contributing input cells is $9 : w_1/2$. Considering that w_1 is hundreds to thousands in a feature map, a distributed LSTM layer has a much larger sensing area than convolutional layers.

(3) Each kernel in a convolutional layer has a sensing area covering multiple rows but each unit in a distributed LSTM layer works on a single row. Therefore, a convolutional layer has a better feature extraction vertically.

(4) A convolutional layer can generate a smaller feature map by setting the s_w and s_h greater than one, but a distributed LSTM layer cannot. Therefore, in a deep neural network, a convolutional layer is useful to narrow the feature map to enlarge the equivalent sensing area for the following layers and save their floating-point operations.

(5) We can combine the convolutional layer and distributed LSTM layer layers in a deep neural network, and the new network should take fewer calculations but has an enhanced ability to extract context features in horizontal.



(a) Convolutional Layer (b) Distributed LSTM layer

Figure 1: Comparison of convolutional layer and distributed LSTM layer schemes

III. Proposed Network

This section introduces the proposed neural network. The network targets to solve a sequential regression problem whose input is a multi-layer feature and whose output is a row vector that has the same width of the input. The network contains three (1) a CNN based local feature encoder, (2) a CNN and LSTM based feature processor, and (3) a CNN based output decoder. Figure 2 presents an overview of network architecture.

A. Local feature encoder

The local feature encoder is a group of convolutional layers designed to extract local features and narrow the feature map size. Commonly a CNN based encoder cascades several convolutional layers with small kernels into blocks because those blocks have fewer parameters to train but work better in non-linear feature extraction. Those blocks are obvious in FCN [7], SegNet [8], and StixelNet [9]. In our work, we cascade six convolutional layers with a $3 \times 3 \times 64$ kernel and group them two by two. The first layer in each group is configured to $stride = (2, 2)$ and the second layer is configured to $stride = (1, 1)$. Therefore, in each group the size of output tensor is half the size of its input tensor in both in both axes. Eventually an input tensor of size $w_1 \times h_1 \times d_1$ will result in a $\frac{w_1}{8} \times \frac{h_1}{8} \times 64$ output tensor.

B. Feature Processor

The feature processor is designed to extract context features over the entire feature map. It is built by several blocks of convolutional layers and distributed LSTM layers. Each block includes a convolutional layer of $3 \times 3 \times 64$ kernel and a distributed LSTM layer of 64 memory states. In the feature processor blocks, there is no pooling layer added so that the input and output tensor have the same size.

C. Output decoder

The output decoder is designed to decode the output tensor of the feature processor to a final row vector. The decoder upsamples the feature map in horizontal but further narrow the feature map in vertical. Since our decoder concentrates on the upper boundary of the drivable region that connects to

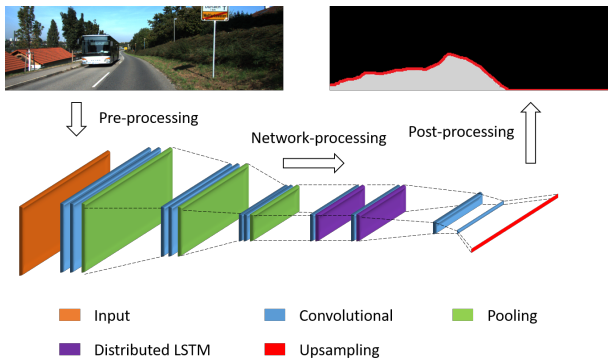


Figure 2: Diagram of the proposed network

the bottom of image frame, it has fewer layers, parameters and calculations than the decoder in SegNet. The decoder includes several convolutional layers, and an upsampling layer. The output size is $w_1 \times 1 \times 1$.

IV. Network Implementation, Training and Testing

In this section, we evaluate the performance of the proposed network using KITTI road benchmark. We first introduce the benchmark and then present our solution using the proposed network. We also present the training scheme for the training set and test result from the testing set.

A. KITTI road benchmark

KITTI road benchmark is a widely used benchmark for drivable region segmentation. It contains 290 training samples and 289 testing samples recorded in a real driving scenario. The size of camera frames in the benchmark is 1242×375 or 1224×370. The drivable region is manually labeled in a binary map of the same size. In the KITTI road benchmark, F1-score (F1) and average precision (AP) are the main metrics to evaluate the accuracy of road segmentation solutions. Other metrics are precision (PRE), recall (REC), false positive rate (FPR), false negative rate (FNR) and running time. The metrics are calculated as in (4) - (9) where TP, TN, FP, FN denote true positive, true negative, false positive, and false negative.

$$PRE = \frac{TP}{TP + FP} \tag{4}$$

$$REC = \frac{TP}{TP + FN} \tag{5}$$

$$FPR = \frac{FP}{TP + FP} \tag{6}$$

$$FNR = \frac{FN}{TP + FN} \tag{7}$$

$$F1 = \frac{2 \cdot PRE \cdot REC}{PRE + REC} \tag{8}$$

$$AP = \frac{TP + TN}{TP + FP + TN + FN} \tag{9}$$

Table I: Detailed Network Blocks

Layer	Kernel $(w_1 \times h_1 \times d_1)$	Input $(w \times h \times d)$	Output $(w_2 \times h_2 \times d_1)$
Input	–	$600 \times 160 \times 5$	–
Conv1 s=2	$3 \times 3 \times 64$	$600 \times 160 \times 5$	$600 \times 160 \times 64$
Conv2	$3 \times 3 \times 64$	$600 \times 160 \times 64$	$600 \times 160 \times 64$
Conv3 s=2	$3 \times 3 \times 64$	$300 \times 80 \times 64$	$300 \times 80 \times 64$
Conv4	$3 \times 3 \times 64$	$300 \times 80 \times 64$	$300 \times 80 \times 64$
Conv5 s=2	$3 \times 3 \times 64$	$150 \times 40 \times 64$	$150 \times 40 \times 64$
Conv6	$3 \times 3 \times 64$	$150 \times 40 \times 64$	$150 \times 40 \times 64$
Conv7	$3 \times 3 \times 64$	$75 \times 20 \times 64$	$75 \times 20 \times 64$
D-LSTM 1	64	$75 \times 20 \times 64$	$75 \times 20 \times 64$
Conv8	$3 \times 3 \times 64$	$75 \times 20 \times 64$	$75 \times 20 \times 64$
D-LSTM 2	64	$75 \times 20 \times 64$	$75 \times 20 \times 64$
Conv9	$1 \times 5 \times 64$	$75 \times 20 \times 64$	$75 \times 4 \times 64$
Conv10	$1 \times 4 \times 1$	$75 \times 4 \times 64$	$75 \times 1 \times 1$
Up-sample	$8 \times 1 \times 1$	$75 \times 1 \times 1$	$600 \times 1 \times 1$
Output	–	–	$600 \times 1 \times 1$

B. Scheme of road segmentation

1) Pre-processing: In pre-processing, we apply a pyramid approach that generates two input tensors from each camera frame. The first input tensor focus on near-range segmentation and the second one focus on far-range segmentation. For the first one, the camera frame is scaled to size 600×160. For the second input tensor, however, the camera frame is cropped to size 600×160 in the center. Those two images are then fed into the tensors together with their horizontal and vertical indexes in their own image coordinate. The pyramid scheme narrows the difference between local features in the near range and far range. It also enlarges the training set for better parameter tuning. After pre-processing, those two input tensors of size $600 \times 160 \times 5$ are processed by two network instances in parallel.

2) Network implementation: The proposed network is implemented in the Keras platform with TensorFlow backend as described in Section III. The details of each layer are presented in Table I. The network has 348,801 parameters and takes 3.45 billion floating-point operations to process each input tensor. Table II compares the proposed network with related works on accuracy, network parameters, floating-point operations and running time for each camera frame. The proposed network has only 24% parameters and takes 1.36% floating-point operations when comparing to SegNet.

3) post-processing: After processing two input tensors in the network, we now have results as two row vectors. The elements in the row vector denote the vertical position of the drivable region boundary in the corresponding column. To transform the row vectors to a drivable region on the camera frame, we draw a polygon that connects corresponding vertices and the bottom line. At last, we scale the first result map to the original size and overwrite the center pixels using the second result map as illustrated in Figure 3.

Table II: Comparison between networks on the KITTI [15] road segmentation challenge: F1-score (F1), average precision (AP), precision (PRE), recall (REC), false positive rate (FPR), false negative rate (FNR), number of parameters (Para), floating operations (FL-OPs) and runtime

Method	F1	AP	PRE	REC	FPR	FNR	Para	FLOPs	Runtime
ours	89.08%	91.60%	88.12%	90.06%	6.69%	9.94%	0.35M	6.9B	16 ms
ours without LSTM	81.84%	73.27%	81.66%	82.02%	10.15%	17.98%	0.36M	7.0B	16 ms
MAP [11]	87.80 %	89.96%	86.01%	89.66%	8.04%	10.34%	457.43M	7.15B	280ms
StixelNet [9]	89.12%	81.23%	85.80%	92.71%	8.45%	7.29%	6.82M	43.0B	1s
Up-Conv-Poly [10]	93.83%	90.47%	94.00%	93.67%	3.29%	6.33%	19.44M	31.5B	83ms

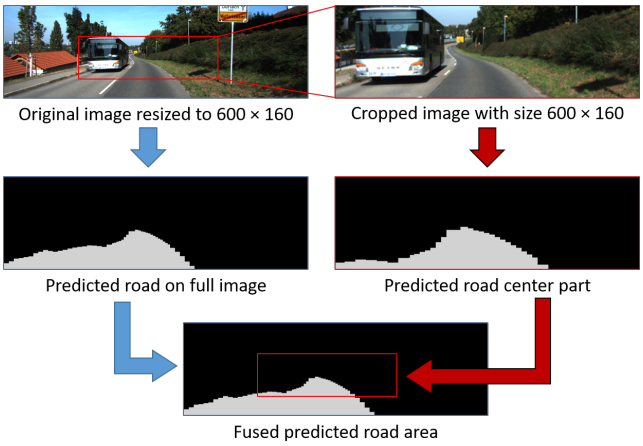


Figure 3: An illustration of the pyramid prediction scheme

Figure 4: Typical road segmentation results

C. Network training and testing

The proposed network is trained in KITTI road training set. There are 289 training samples and the image sizes range from 1242×375 to 1224×370. The ground truth of each sample is formatted as a binary image with the same size. To augment the training set we scale the sample images by 0.5 and 1.0 and then use a sliding window of size 600×160 to capture the images and labels. The stride of the sliding window is 60 pixels in horizontal and 20 pixels in vertical. Finally, $20,808$ samples are generated. Those samples are separated into a $20,000$ sample training set and an 808 sample validating set. We also add Gaussian noise to the input data with a standard deviation of 0.02% for additional diversity. Adam [16] is a gradient descent based optimizer that adjusts learning rate on each neuron according to the estimation of lower-order moments of the gradients. We choose this optimizer because it accelerates the converging process in the beginning and slows it down near optimum. During training, the input batch size is set to 100 and reference learning rage is set to $1e^{-5}$. After training 80 epochs we get 0.0185 mean average error on validation data.

We then apply the network on the KITTI road testing set and submit to the benchmark server. The result shows that the proposed solution achieved 89.08% in F1-score and 91.60% in average precision (AP), which are comparable to the existing approaches. In Table II our work is compared to the related solutions listed on the KITTI road benchmark. It shows that our work has a similar F1-score and AP to other works but a higher precision and lower false positive rate. The proposed network has fewer parameters to train and takes less floating-point operations to process. When testing on an NVidia GTX 950M GPU, the proposed solution achieved 16 ms/frame network runtime. We also trained a pure-CNN network by replacing the distributed LSTM layers with 3×3 convolutional layers. The accuracy of the pure-CNN solution is worse than that of CNN-LSTM solution, as shown in Table II.

Figure 4 shows the typical result of the proposed road detector. Green pixels are true positives, red pixes are false negatives, and blue pixels are false positives. It is obvious that most of the road surface is detected, and obstacles such as vehicles and railways are separated to avoid collisions. False negatives occasionally appear at road-vehicle and road-sidewalk interfaces , which is not a safety concern for automated driving. However, false positive on the sidewalks needs further corrections.

V. Conclusion

In this paper, we compare the convolutional layer and distributed LSTM layer and demonstrate the advantages of combing the CNN and LSTM structures for spatial feature map processing. We also propose a neural network to evaluate its performance. The test result on the KITTI road benchmark shows that our solution achieves 89.08% in F1-score and 91.60% in average precision. However, the image-based road segmentation is subjected to illumination conditions. Shadows, blurs, and ambiguous textures can cause false positives and false negatives. In future work, a fusion of multiple sensors including camera, LiDAR and radar will be applied to improve the road segmentation as well as object detection.

VI. Acknowledgement

This work is supported by U.S. NSF Grant CNS-1626236.

REFERENCES

[1] Aharon Bar Hillel, Ronen Lerner, Dan Levi, and Guy Raz. Recent progress in road and lane detection: a survey. *Machine vision and applications*, 25(3):727–745, 2014.

[2] Alexandre Alahi, Kratarth Goel, Vignesh Ramanathan, Alexandre Robicquet, Li Fei-Fei, and Silvio Savarese. Social lstm: Human trajectory prediction in crowded spaces. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 961–971, 2016.

[3] Yecheng Lyu, Lin Bai, and Xinming Huang. Real-time road segmentation using lidar data processing on an fpga. In *Circuits and Systems (ISCAS), 2018 IEEE International Symposium on*, pages 1–5. IEEE, 2018.

[4] Yecheng Lyu, Lin Bai, and Xinming Huang. Chipnet: Real-time lidar processing for drivable region segmentation on an fpga. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2018.

[5] Hunjae Yoo, Ukil Yang, and Kwanghoon Sohn. Gradient-enhancing conversion for illumination-robust lane detection. *IEEE Transactions on Intelligent Transportation Systems*, 14(3):1083–1094, 2013.

[6] Hunjae Yoo, Ukil Yang, and Kwanghoon Sohn. Gradient-enhancing conversion for illumination-robust lane detection. *IEEE Transactions on Intelligent Transportation Systems*, 14(3):1083–1094, 2013.

[7] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440, 2015.

[8] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *arXiv preprint arXiv:1511.00561*, 2015.

[9] Dan Levi, Noa Garnett, Ethan Fetaya, and Israel Herzlyia. Stixelnet: A deep convolutional network for obstacle detection and road segmentation. In *BMVC*, pages 109–1, 2015.

[10] Gabriel Leivas Oliveira, Wolfram Burgard, and Thomas Brox. Efficient deep methods for monocular road segmentation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2016)*, 2016.

[11] Ankit Laddha, Mehmet Kemal Kocamaz, Luis E Navarro-Serment, and Martial Hebert. Map-supervised road detection. In *Intelligent Vehicles Symposium (IV), 2016 IEEE*, pages 118–123. IEEE, 2016.

[12] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1705.04861, 2017.

[13] François Chollet. Xception: Deep learning with depthwise separable convolutions. *CoRR*, abs/1610.02357, 2016.

[14] Yuteng Zhou, Yecheng Lyu, and Xinming Huang. Roadnet: an 80mw hardware accelerator for road detection. *IEEE Embedded Systems Letters*, 2018.

[15] Jannik Fritsch, Tobias Kuhnl, and Andreas Geiger. A new performance measure and evaluation benchmark for road detection algorithms. In *Intelligent Transportation Systems-(ITSC), 2013 16th International IEEE Conference on*, pages 1693–1700. IEEE, 2013.

[16] Diederik P. Kingma and Jimmy Lei Ba. Adam: a Method for Stochastic Optimization. *International Conference on Learning Representations 2015*, pages 1–15, 2015.