Real-Time Road Segmentation Using LiDAR Data Processing on an FPGA

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—This paper presents the FPGA design of a convolutional neural network (CNN) based road segmentation algorithm for real-time processing of LiDAR data. For autonomous vehicles, it is important to perform road segmentation and obstacle detection such that the drivable region can be identified for path planning. Traditional road segmentation algorithms are mainly based on image data from cameras, which is subjected to the light condition as well as the quality of road markings. LiDAR sensor can obtain the 3D geometry information of the vehicle surroundings with very high accuracy. However, it is a computational challenge to process a large amount of LiDAR data at real-time. In this work, a convolutional neural network model is proposed and trained to perform semantic segmentation using the LiDAR sensor data. Furthermore, an efficient hardware design is implemented on the FPGA that can process each LiDAR scan in 16.9ms, which is much faster than the previous works. Evaluated using KITTI road benchmarks, the proposed solution achieves high accuracy of road segmentation.

Index Terms—Autonomous vehicle, road segmentation, CNN, LiDAR, FPGA

I. INTRODUCTION

In recent years, we have witnessed a strong increase of research interests on advanced driver assistance systems (ADAS) and autonomous vehicles. While fully autonomous driving might still be years away, there are many recent research on traffic scene perception and its implementations on various platforms. The traffic scene perception task can be separated into two sub-tasks: object detection and road/lane detection. Object detection includes vehicle detection [1] [2] [3] [4] [5], pedestrian detection [6] [3] [4] and traffic light/sign detection [7] [8] [9] [10] [11], while road/lane detection includes road marking detection [12] [13] [14], lane detection [15] [16] [17] and road segmentation [18] [19] [16]. In this work, we are primarily concentrating on road segmentation, since it is a fundamental component of automated driving that provides the drivable region for the vehicle’s next movement.

Many sensing modalities have been used for road segmentation. Monocular vision [16] [5] [20] and stereo vision [21] [22] are mostly used because cameras are low-cost and have a similar view to human eyes. However, considering road appearance diversity, image clarity issues, poor visibility conditions [23], image-based feature describers are often difficult to generate and easy to fail. In contrast of passive sensors such as cameras, light detection and ranging (LiDAR) actively emits laser beams and measures the distance from the reflection by time of flight (TOF). Therefore, LiDAR is robust to environmental illumination. Several recent works have studied road segmentation based on LiDAR information or the combination of LiDAR and camera data [24], [25].

For the applications of autonomous vehicles, both real-time performance and power consumption need to be considered [26]. Graphic Processing Unit (GPU) is a popular platform for parallel processing, but power consumption is usually high. FPGA suits to the condition with limited power supply, such as an autonomous vehicle. Moreover, FPGA can be developed as a customized integrated circuit that can perform massive parallel processing and data communications on-chip. Hereby, we propose to target the LiDAR based road segmentation algorithm on an FPGA as a real-time low-power embedded system.

In this paper, the problem of road segmentation is framed as a semantic segmentation task in spherical image using a deep neural network. Instead of an encoder-decoder structure often implemented in traditional neural networks, a block containing a convolutional layer and a non-linear layer is cascaded twelve times so that multiplexing can be applied on the processing blocks on-chip. The proposed solution is evaluated on KITTI benchmarks and achieve satisfactory result. The rest of paper is organized as follows. Section II introduces the related work of road perception problem. The proposed convolutional neural network (CNN) structure and its performance on KITTI benchmarks are presented in Section III. Section IV presents the FPGA design hardware architecture and implementation results. Finally Section V concludes the paper.

II. RELATED WORK

Road segmentation has been studied with different sensors and algorithms over the past decade. In the early years, researchers used manually designed feature descriptors to separate the road from others. At that time, camera was the major sensor and features were often generated based on the illumination and shape from images [14] [16] [27], which led to low accuracy and the performance variations from different light conditions and road scenes. Recently, two major techniques have been investigated to overcome the shortcomings of manually selected features in images.
One is to use machine learning to design a complex and robust feature descriptor, such as CNN \cite{28} \cite{4} \cite{20} and conditional random field (CRF) \cite{29}. The other is to use intensity invariant sensor or multi-sensor fusion instead of camera to obtain a more robust descriptor. A popular intensity invariant sensor is LiDAR \cite{30}. There were also research works trying to combine those two and apply machine learning to data processing. Several results showed high accuracy, but their processing time is too long to be employed for real-time applications \cite{25} \cite{24}. For autonomous driving, road segmentation must be implemented on real-time embedded platforms such as FPGA, application-specific integrated circuit (ASIC), or a mobile CPU/GPU processor. A neural network was proposed in \cite{31} to detect lane markers on the road and had the run-time of 2.5Hz on TK1 mobile GPU platform. Similarly, research work in \cite{32} proposed a neural network to segment multiple objects including vehicle, pedestrian and pavement and achieved 10Hz run time at the resolution of 480-by-320 pixels on TX1 GPU platform. In \cite{15} and \cite{33}, FPGA based solutions are proposed for lane detection and resulted 60Hz and 550Hz processing speed, respectively.

III. ALGORITHMS DESIGN

The goal of road segmentation is to label the drivable region, also called free space. The input data comes from different sensors such as camera, LiDAR, GPS and IMU. The output are usually presented as area on the top-view or labeled pixels on camera view. In this paper, we choose LiDAR data as input, a deep neural network as the processor, and top-view predictions as the main output to evaluate the road segmentation performance. Results on camera view are also presented for better visualization. The proposed algorithm has the following three steps: pre-processing, neural network processing and post-processing.

A. Pre-processing

During pre-processing, input data points are arranged and projected into a 3-D blob with $M$ by $N$ tensors and $C$ channels so that the tensor can flow through the layers in the neural network to produce an output. A tensor refers to a specific view in the real world. There are four types of views available for the autonomous driving task: image view (also known as camera view), top view (also known as bird eye view), cylindrical view and spherical view. Image view and top view are commonly choices, because in this two views LiDAR data can be fused with camera data and those views are natural to human eyes. However, LiDAR points are sparse in those views. Statistically, LiDAR points covers only 4% of pixels in image view and 5.6% on top-view. That means majority inputs into the neural network are zero input leading to waste of computing resource. Cylindrical view and Spherical view match the LiDAR sensing scheme and data points can cover up to 91% pixels on the map. Hereby we choose spherical view as the projection scheme. The resolution of polar angle $\theta$ and azimuthal angle $\varphi$ are chosen based on the LiDAR resolution. In this work, all 64 rows are included in vertical. While in horizontal LiDAR points are grouped by 0.4$^\circ$ which doubles the designed resolution of LiDAR to minimum the number of cells without LiDAR measurements. The input blob has 256 columns and FOV is shifting to augment training data.

Although spherical view is chosen for data projection, we can still add additional feature channels from other views to improve the accuracy of the trained neural network. Here we select sixteen channels, the first 7 channels come from the LiDAR point which has the lowest altitude in the cell, the next 7 channels come from the LiDAR point which has the highest altitude. The 7 channels are location of measured points in Cartesian coordinate ($x, y, z$), location of measured points in spherical coordinate ($\theta, \varphi, r$), and reflection intensity of measured points ($H$). The other 2 channels are the location of cell on the 2D map ($i, j$).

B. Neural network processing

In autonomous driving, traffic scene perception is often implemented on embedded systems. In consideration of limited computational resource in an embedded system, we proposed
a new network architecture that minimize the GPU and FPGA memory by multiplexing the blob memory. The architecture is shown in Figure 4. Except for the first and last convolutional layers, those 9 layers in between are constructed using the same structure. Each repetitive structure includes a convolutional layer and an activation layer. The convolutional layer is built with 64 filters and each filter has a $5 \times 5$ kernel with stride size of 1 and padding size of 2. The stride and padding settings make output of the convolutional has the same size as the input. Rectified linear unit is chosen as the activation function for fast training. Two drop-out layers are added after 6th block and 10th block in training phase to accelerate convergence. It can be seen that there is no pooling layers and all blobs have the same size except for the input blob and score blob. Therefore all internal results can be stored in the same memory space directory without allocation or reshaping the blob. We choose $5 \times 5$ convolutional kernel size and 11 convolutional layers from our experimental results that this settings is a good trade-off between CNN performance and resource usage.

C. Post-processing

In post-processing, results obtained from the neural network are projected back to targeted views, i.e. camera view and top view, for performance validation. The challenge of the post-processing is that the points in the output of neural network are non-uniformly distributed on the target view after projection. Traditional image processing methods, such as dilation, erosion, closing and opening, are not able to generate a filled area with smoothed contour. In our post-processing step, contour of the drivable area is firstly determined and then the region within the contour is marked as the segmentation results on target view. Figure 5 shows an example of the road segmentation results.

To determine the contour of the drivable area, the furthest points in each angle $\theta$, which is corresponding to the each column of the neural network output, are selected and projected onto the target view. Subsequently, a polyline is drawn along those furthest points on all angles on the target view. The polyline graph becomes a polygon if we add a straight line at the bottom. The polygon is then treated as contour of drivable area and filled up with semantic pixel labels.

D. Training and evaluation on KITTI road benchmark

To evaluate the performance of the proposed approach, we train the network on KITTI road/lane detection dataset. As described in [34], maximum F1 score ($F_{\text{max}}$) and average precision (AP) are the key measurement to evaluate the performance of road perception algorithms. $F_{\text{max}}$ provides the insight of an algorithm’s optimal performance, while AP indicates its average performance. In Table I, we compared our proposed approach with several results published recently. It shows that our proposed approach has comparable performance but uses significantly less processing time. The actual processing time of the neural network implemented on the FPGA is about 16.9ms. Since most of the execution times listed in Table 1 were from various GPU platforms, we also evaluate our algorithm on a K20 GPU using MATLAB on Caffe and the total processing time is about 120ms, including pre-processing, neural network, post-processing, and visualization.

| Name              | $F_{\text{max}}$ | AP   | run time |
|-------------------|------------------|------|---------|
| This work on FPGA | 91.79%           | 84.76% | 16.9ms  |
| HybridCRF [25]    | 90.81%           | 84.79% | 1500ms  |
| LidarHisto [35]   | 90.67%           | 84.79% | 100ms   |
| MixedCRF          | 90.67%           | 84.79% | 6000ms  |
| FusedCRF [24]     | 88.25%           | 79.24% | 2000ms  |
| RES3D-Velo [36]   | 86.58%           | 78.34% | 360ms   |

IV. HARDWARE ARCHITECTURE

As described in Section III, we organize the LiDAR data into an image map with 16 channels in the size of $256 \times 64$. The block diagram of the convolutional layer architecture is shown in Figure 6. The same convolutional unit is used repetitively. There are totally 64 memories to store intermediate feature map and each memory size is 256k bits. The large 3D convolution can be broken into 64 parallel 2D convolutions, each with 2 filters, followed by an adder tree to generate the feature map.
Figure 6: Hardware architecture of the implementation of convolution layer.

Figure 7: An illustration of the zero-padding in the RAM

A. Zero padding generation

Zero padding helps to control the size of feature maps and to reserve the boundary information of the input images in convolution operation. Since the input images are transmitted without padding, a special dual-port RAM is designed for the convenience of the next stage convolution. As shown in Figure 7, each slot of RAM represents one column of the input image. The padded zeros are stored in the block RAM in advance. Control logic is used to store each pixel into proper memory location. On the other side of the memory, a scanning circuit reads out data from this RAM pixel by pixel.

B. 2D convolution

2D convolution is implemented in conjunction with a line buffer which consists of 4 lines and 5 additional registers. As demonstrated in Figure 8, it outputs $5 \times 5$ pixel window in parallel for the multiplication with the weight matrix using 25 multipliers. A highly pipelined adder tree follows the multiplication to compute the sum.

C. Control logic

Because of the large RAM consumption for images with zero padding and feature maps, a loop-based control is proposed. Each 2D convolution could generate 2 feature maps. One finite state machine is used to generate 64 feature maps in 32 loops, reusing the block RAM for images with padding. To achieve 11 layers of convolution, another finite state machine is implemented for loop controlling. Therefore, completing the 11 fully convolutional layers with each depth of 64 requires to perform the 2D convolutions 352 times.

D. Implementation results

We implement this fully convolutional network on Xilinx UltraScale XCKU115 FPGA. The targeted operating frequency is set to 350MHz. Each 2D convolution takes about 18,000 clock cycles. It takes about 16.9ms to complete all 11 convolutional layers, each with filter depth of 64 (except for depth of 16 in the first layer and the depth of 2 in the last layer). Since LiDAR normally scans at 10Hz, this FPGA implementation fulfills the requirement of real-time processing. When tested on the Intel Xeon CPU E5-2687W v3, the processing time is about 500ms. Therefore, the FPGA implementation gains the speedup factor of 30 over CPU. The resource usage of the FPGA implementation is listed in Table II.

V. Conclusions and Future Work

In this paper, we propose a neural network based approach for road segmentation using LiDAR data. The neural network is trained with KITTI road/lane detection dataset and evaluated on its test benchmark. Moreover, the proposed fully connected neural network is implemented on an FPGA for real-time low-power processing, which results the processing time of only 16.9ms for each LiDAR scan. The implementation consumes a large amount of FPGA on-chip memory.

For future work, we are considering using the external DDR4 SDRAM to store feature maps. We also notice during testing that sidewalk and railway with same altitude as road pavement contributes to the majority of false positive. Fusion of LiDAR and camera data is needed to further improve the accuracy.
