ChipNet: Real-Time LiDAR Processing for Drivable Region Segmentation on an FPGA

Yecheng Lyu, Lin Bai and Xinming Huang Senior Member, IEEE

Abstract—This paper presents a field-programmable gate array (FPGA) design of a segmentation algorithm based on convolutional neural network (CNN) that can process light detection and ranging (LiDAR) data in real-time. For autonomous vehicles, drivable region segmentation is an essential step that sets up the static constraints for planning tasks. Traditional drivable region segmentation algorithms are mostly developed on camera data, which is subjected to the light condition as well as the quality of road markings. LiDAR sensors can obtain the 3D geometry information of the vehicle surroundings with high precision. However, it is a computational challenge to process a large amount of LiDAR data in real-time. In this paper, a convolutional neural network model is proposed and trained to perform semantic segmentation using data from the LiDAR sensor. An efficient hardware architecture is proposed and implemented on an FPGA that can process each LiDAR scan in 17.59 ms, which is much faster than the previous works. Evaluated using Ford and KITTI road detection benchmarks, the proposed solution achieves both high accuracy in performance and real-time processing in speed.

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

I. INTRODUCTION

In recent years, we have witnessed a strong increase of research interests on autonomous vehicles [22] [8]. Since the DAPRA Urban Challenge in 2007, automated driving technology has grown rapidly from research experiments to commercial vehicle prototypes owing to the explosive progress in the fields of artificial intelligence and machine learning. As an important component of automated driving system, it is critical to conduct research on traffic scene perception and its implementations on hardware platforms.

In traffic scene perception algorithms, the detecting and tracking components are aimed to perceive the surroundings and to set the constraints for planning and control tasks. Based on the object types, the task of traffic scene perception can be classified into three sub-tasks: (1) road perception including drivable region segmentation and lane detection, (2) obstacle detection/tracking, and (3) traffic sign/signal detection. In road perception, drivable region segmentation scans the front area and searches for the drivable region, while lane detection narrows the region of planning to the ego-lane if lane markers are visible. Obstacle detection and tracking focus on the moving objects such as vehicles, pedestrians, cyclists and animals, and measure their locations, dimensions and speed to avoid a collision. Traffic sign/signal detection looks for traffic signs and traffic lights to perceive additional constraints for planning tasks. As a critical component of an automated driving system, drivable region segmentation provides fundamental knowledge of driving environment. Drivable region segmentation solutions are required to perceive a wide range of view, generate accurate results, and respond in real-time. However, road scenes are complicated. As described in [18], road scenes have three types of diversities: (1) appearance diversity due to changing shapes of lane markers and camera lens distortion, (2) clarity diversity due to occlusions and illumination, and (3) visibility condition diversity due to weather condition.

Many sensing modalities have been used for drivable region segmentation. Vision modalities [29] [2] [12] are frequently applied on drivable region segmentation for two major reasons: (1) Vision modality is similar to human visual system and most road markers have features in the visual domain, and (2) As a passive sensor, visual camera provides high-resolution data with rich features. By implementing multiple cameras, stereo vision [5] can provide depth information for drivable region segmentation. However, due to the diversity in road scene, it is difficult to design a feature descriptor that handles all visual cases and light conditions. In addition, Shen et al. proposed a series of algorithms to cluster super-pixels that could improve vision based semantic segmentation [29] [31] [32] [9] [28] [30].

Light Detection And Ranging (LiDAR) is another major modality widely used by autonomous vehicles. By actively emitting laser beams and measuring the 3D geometry around the vehicle using Time of Flight (ToF), LiDAR can provide a few million geometry points per frame with centimeter accuracy. In addition, LiDAR is not subjected to environmental illumination. However, compared to vision modalities, LiDAR points are sparse and do not contain any visual features employed in traditional vision based algorithms. Several recent works have studied traffic scene perception involving LiDAR modality and have proposed several schemes for data arrangement, feature extraction and sensor fusion with monocular vision. Much in depth studies are needed on LiDAR data arrangement and feature extraction to build an accurate and efficient LiDAR based drivable region segmentator.

In the past decades, drivable region segmentation has been studied with different sensors and methodologies. A general solution consists of four components: pre-processing, feature extraction, detection and post-processing. Pre-processing includes noise removal, data sampling and transformation. Feature extraction encodes local features such as color, edge and texture from pre-processed data. Detection applies manually
defined or machine learning based models to detect road area or lane boundaries. Lastly, post-processing suppresses candidates to provide final results. In traditional computer vision algorithms, those four steps are totally separated and features extracted in the solution are often describable. However, manually defined features and detectors only work well in normal conditions but cannot handle much variations on the road. Machine learning especially convolutional neural network (CNN) based algorithms combine feature extraction and detection together. Pre-processed data are fed into a well-structured CNN with millions of parameters. Despite that features and detectors are hardly describable visually, machine learning based road perception algorithms have significant advantages in accuracy when compared with traditional computer vision based approach. Moreover, with the support of graphics processing unit (GPU) and other parallel computing devices such as field-programmable gate array (FPGA) and application-specific integrated circuit (ASIC), CNN-based algorithm can be accelerated for real-time applications.

For autonomous vehicles, both real-time processing and low power consumption are desirable. GPU devices are popular for parallel processing, but usually consume too much power. Currently only one or two GPU devices can be installed in a vehicle due to the limited power supply while tens of perception and planning tasks need to be processed on the GPUs simultaneously. FPGAs are low-power devices that are suitable for embedded systems with power constraints. Moreover, an FPGA can be developed as a customized integrated circuit that can perform massive parallel processing and data communications on-chip. Hereby, FPGA is our chosen platform that fits both computational capability requirement and power consumption constraint.

In this paper, we present ChipNet as a CNN-based algorithm and its FPGA implementation for real-time LiDAR data processing. The contributions of our work can be summarized as follows. (1) We introduce a new data organizing and sampling method in spherical coordinate that improves the usage of LiDAR points and creates a dense input tensor for CNN. (2) We propose an efficient convolution block for CNN that is both hardware friendly and extendable. (3) The proposed approach of drivable region segmentation results the state-of-art accuracy when evaluated using Ford dataset and KITTI benchmark. We also label the Ford dataset for training and evaluation. (4) An efficient and flexible 3D convolution module is designed and implemented on an FPGA, which can achieve real-time processing speed with limited hardware resource and power usage.

The rest of the paper is organized as follows. Section II introduces the related works on road perception task. The proposed drivable region segmentation algorithm is described and its performance on benchmarks are presented in Section III. Section IV presents the FPGA architecture and hardware implementation results. Finally, Section V concludes the paper.

II. RELATED WORK

LiDAR data arrangement: There exists various methods of LiDAR data arrangement on traffic scene perception. \[1\], \[3\] and \[24\] projected the LiDAR point cloud to image view and applied manually defined features based on evaluation measurements with image patches. \[15\] and \[14\] also projected point cloud to image view followed by feature extraction using histogram. \[13\] and \[38\] created a dense depth map from point cloud and then combined the map with the camera data for their machine learning based road boundary detector. Similarly, \[6\] transformed point cloud into both image and top views and then combined with camera data for sensor fusion using a CNN. In addition, \[41\] and \[23\] directly processed sparse LiDAR data in world coordinate using convolutional neural network. LoDNN \[3\] organized the point cloud into a top view and then fed it into a CNN to generate a heat map representing the possibility of drivable region in each \(0.1m \times 0.1m\) cell. Beside road perception, several research works proposed using CNN for LiDAR-based vehicle detection \[41\] \[6\]. To overcome the shortage of training samples, data augmentation and coarse labeling methods were proposed to enlarge the dataset. \[41\] augmented training data by rotating and translating LiDAR points together with ground truth. StixelNet \[12\] used LiDAR points to generate coarse labeling automatically for pre-training.

CNN for road perception: Convolutional neural networks have become an active approach for the task of road perception. Starting from Fully Convolutional Network (FCN) \[25\], various network structures have been proposed to provide accurate road detection and segmentation. Segnet \[2\] introduced an encoder-decoder scheme to separate feature extractor and detector components. Furthermore, \[2\] built connections between layers in the encoders and decoders that improved the training of those layers closer to the input. \[26\] followed the encoder-decoder scheme and perceived near range and far
range in separate branches that resulted in an increased accuracy of vision based segmentation. RBNet [7] also followed the encoder-decoder scheme but connected all encoder layer outputs to the decoders. Most recently, CNN has also been introduced to LiDAR based road segmentation. LoDNN [3], VoxelNet [41] and [6] are advanced works on LiDAR based perception.

Embedded platforms for road perception: For situations like autonomous driving or advanced driver assistance system (ADAS), the processing time of road perception algorithms must fulfill the real-time requirement, and thus are often implemented on embedded platforms such as FPGA, ASIC or a mobile CPU/GPU processor. [19] deployed a neural network on Jetson TK1 mobile GPU platform. It detected lane markers based on images and achieved 2.5 Hz running speed. Similarly, the neural network proposed by [36] was able to segment multiple objects including vehicles, pedestrian and pavements at 10 Hz with image resolution of 320p on TX1 GPU platform. In [40] and [36], two FPGA based lane detection solutions were proposed and their processing time were at 60 Hz and 550 Hz, respectively.

III. Algorithm Design

In this work, a hardware friendly and extendable convolutional neural network is proposed to segment drivable region from LiDAR data. In this section, we first introduce the LiDAR data preparation method as pre-processing of the CNN. Next, the proposed network architecture ChipNet is described in detail. Furthermore, we introduce a simulated quantization scheme for CNN that transforms floating-point to fixed-point operations, and thus speeds up the processing on hardware considerably. Finally, post-processing is developed to generate a decision map denoting the drivable regions from the CNN output. The proposed solution is evaluated on Ford Campus Vision and LiDAR dataset and KITTI road benchmark. The performance results are presented towards the end of this section.

A. LiDAR data preparation

Typically a LiDAR device places a number of laser scanners vertically and rotate them azimuthally to scan the surrounding obstacles. Suppose a LiDAR device that contains \( N \) scanners, measures \( M \) points per second and rotates at \( R \) rpm, then it generates \( \frac{R}{60} \) frames per second with \( \frac{0.6M}{\phi} \) measure points per frame at an azimuthal resolution of \( \frac{360 \times 60}{60M} \). The polar resolution is \( \frac{\phi}{2} \) where \( \phi \) denotes the vertical field of view. For example, the HDL-64E LiDAR used in KITTI road benchmark [11] has 64 scan channels and emits 1.33 million points per second. By rotating at 600 rpm it updates 10 frames per second with 0.133 million measurement points per frame at 0.17° azimuthal resolution. By focusing on a 26.90° vertical field of view, the polar resolution is 0.42°. In practice, the LiDAR sensor occasionally generates void measure points when the laser beam emits to a low reflective surface.

A typical frame of data generated in LiDAR modality is a 2D array as shown in Figure 1. In each row, four measurements of the corresponding LiDAR point are listed by columns that are location coordinates \( x, y, z \) of the LiDAR view and laser reflection intensity of the target surface \( r \). By projecting all points to camera view and top-view, as presented in Figure 1 LiDAR point cloud is sparse and has large variations of point density throughout the entire space. Therefore, LiDAR data needs to be organized and re-sampled before being fed to the neural network processor. As mentioned in Section II there is no unified method to LiDAR point cloud data arrangement and sampling view. Table I summarizes several research works that organized LiDAR data in different ways for neural networks. In Table I we can see that most of them divided the 3D space in Cartesian coordinates, but they sampled the point cloud in different views, such as top view [3], [6], front view [41] or 3D view [23]. We also find mostly a high percentage of LiDAR points are encoded in their region of interest (RoI). However, the organized data as the input to neural networks are sparse, which means that the majority of computations in the first few layers of CNN actually deal with zeros. That is very inefficient from computing perspective.

In our work, LiDAR data is organized in spherical view that a LiDAR naturally scans the surroundings, as shown in Figure 3. A region of interest is selected in azimuth \([-45°, 45°]\) and all 64 lines of scan points are involved in segmentation. On each line, scan points are grouped by every 0.5° into cells. In total, all scan points in the RoI fall into a 180 × 64 mesh. We use 0.5° because it is 3 times of LiDAR azimuthal resolution so that in theory at least 2 scan points are grouped in each cell. In practical terms, there are some void scans when the reflect surface is out of range or has low-reflectivity. Input tensor is built with same width and height as the scan point mesh, but contains 14 feature channels. In each cell, the first 7 features come from the point nearest to the scanner, the next 7 features come from the point furthest to the scanner. These 7 features include Cartesian coordinates \( x, y, z \), spherical coordinates \( \theta, \varphi, \rho \) and the laser reflection intensity \( r \).

In Table I LiDAR data preparation in our work is compared with several related works. By sampling LiDAR points in spherical view, our work not only has high LiDAR point usage in RoI but also creates a dense input tensor that improve the accuracy performance and makes the computations in the CNN much more efficient.

B. ChipNet: a hardware friendly and extendable CNN architecture

1) ChipNet convolution block: Each network block contains three branches. The first one is the identity branch that directly copies the input to the output. As analyzed in [17], identity branch contributes the majority of gradient in back-propagation and decreases the chance of gradient vanishing and explosion during training. The second branch is a 3 × 3 convolutional layer with 64 channel outputs. The second branch is aimed to encode local features. The third branch is a dilated 3 × 3 convolutional layer [39] to process features in further pixels but takes less parameters and calculations. As shown in Figure 4 after adding all three branches element-wise, the block equivalents to a 5 × 5 convolutional layer but has a stable gradient in back-propagation and fewer
2) ChipNet network architecture: The overall CNN architecture of ChipNet is shown in Table II. The first layer is a local feature encoder aimed to encode the input LiDAR data into a 64-channel feature tensor. After encoding, the proposed ChipNet convolution block is instantiated repetitively in the network to perform additional encoding and decoding. Since the input and output of the ChipNet block have exactly the same sizes, the neural network can be conveniently extended deeper by adding more layers. In our work, the ChipNet block is instantiated 10 times as a trade-off between segment accuracy and processing latency. For the output layer, a channel-wise mapping is used to generate the final decision map showing the probability of corresponding drivable regions.

Compared to FCN and SegNet, the proposed network is much simpler and more importantly it is extendable. The repetitive network structure is best fitted for hardware reuse in the FPGA design.

3) Simulated Quantization: Fixed-point variables, weights and operations are widely used in FPGA design, which often utilizes less hardware resources and memories and results in higher clock speed, if compared with the floating-point implementations. However, CPU and GPU platforms generally employ floating-point operations that have no quantization error and can generate continuous gradients in the training
session. Practically, we can implement CNNs on an FPGA for low-power embedded application. But we still heavily rely on the high-performance GPUs to train the neural networks in order to generate the parameters and weights, since a GPU machine is capable of storing terabytes of training samples and processing hundreds of threads simultaneously.

However, we cannot simply quantize all variables and parameters of a pre-trained neural network from floating-point into fixed-point. Since quantization is a nonlinear operation, the results are not optimal for the fixed-point neural work. In addition, direct quantization of variables and parameters may result a loss of gradients. Hereby, we propose a simulated quantization method to train a neural network that can produce the optimal parameters in a defined fixed-point form. This is an essential step to prepare a CNN before implement it on an FPGA.

Figure 5: (a) Simulated quantization method as in [20] and (b) the proposed quantization method in this work

**Simulated quantization of weights:** Quantization that we refer here is not just simply an operation of quantizing pre-trained floating-point weights to fixed-point numbers. Additional training is needed after quantizing the weights to avoid negative impact of accuracy. During the training session using back-propagation, however, floating-point weights are preferred because we want to avoid gradient exploding and vanishing. In [20] a simulated quantization approach was proposed in which weights and gradients are stored as floating-point numbers during back-propagation training but the quantized fixed-point numbers are used during forward convolutional operations. The advantage was that the weights and gradients are updated in continuous space so that local optimum due to quantization can be avoided. The disadvantage that several key functions need to be modified to support this method, however it is usually difficult to modify, maintain and distribute customized components in a general machine-learning platform such as TensorFlow.

In our work, a new weight regulator is defined and added to the existing network. The regulator is described in Algorithm 1. The key innovation is that the regulator quantizes the weights during training and the fixed-point numbers are used during forward operations. Meanwhile, the floating-point weights are also stored in the memory that are used when computing the gradients during back-propagation. The stopGradient function is set to 0 during gradient operations. Hence, the new feature is imported to the existing platform as a plug-in rather than a merge request. The proposed quantization algorithm and data flow are shown in Figure 5 in comparison to the simulated quantization in [20].

**Algorithm 1 weight quantization**

Data Weights W, Gradients G
Parameter total_bits=N, fraction_bits=F

1: Fraction_scalar ← $2^F$
2: Upper_bound ← $2^{N-1} - 1$
3: Lower_bound ← $-2^{N-1}$
4: $\hat{W} ← W \times$ Fraction_scalar
5: $\hat{W} ← \text{round}(\hat{W})$
6: $\hat{W} ← \max(\hat{W}, \text{Lower_bound})$
7: $\hat{W} ← \min(\hat{W}, \text{Upper_bound})$
8: $\hat{W} ← \hat{W} / \text{Fraction_scalar}$
9: $W ← W + \text{StopGradient}(\hat{W} - W)$
10: return $\{W, G\}$

**Simulated quantization of variables:** Quantization of variables is very similar to that of weights when applied to the forward operations. Quantization of variables is implemented as a new activation function so that it can be imported as a custom defined function rather than modifying the body of an existing platform. As described in Algorithm 1 if we convert a floating-point number to a N-bit fixed-point number with an F-bit fraction, the operation is to shift to the left by F bits and then round it to a N-bit integer, followed by shifting back F bits to the right. To minimize negative impact to back-propagation training, the gradients are all computed in floating-point. Figure 5 presents the process of variable quantization in forward and back-propagation through a convolutional layer.

**Evaluation of Quantization:** To evaluate the influence of quantization on accuracy, we train the ChipNet without quantization on Ford training set, and then quantized and
fine tuned using the same training set. Both versions of the ChipNet with and without quantization are evaluated on the Ford dataset. The result listed in Table III shows that ChipNet quantized to 18 or more bits has similar performance compared to that without quantization, which indicates that our proposed quantization scheme does not cause accuracy degradation for the convolutional neural networks.

C. Post-processing

The output of network denotes the possibility of drivable region for each cell in spherical view. In post-processing, the output is projected to the top view of a 20-meter wide and 40-meter long area in front of the vehicle. One reason of this projection is that constraints are easy to set up on top view. In addition, the top view matches the output data format in KITTI benchmark [11] so we can compare our results with others reported in the dataset. The post-processing algorithm is described in Algorithm 2. Suppose the possibility threshold of a drivable region is set to THR, then the reference point in each column \( j \) in the network output \( P_{i,j} \) is determined by the nearest LiDAR point in group \( \{ P_{|col=i, p<THR} \} \). After generating the reference points, a contour of the drivable region becomes a polygon that contains all reference points as vertices.

The post-processing scheme is implemented on CPU using GridMap [10], an Eigen based universal grid map management library. The GridMap library stores map data as Eigen matrix and supports iterators for rectangular, circular, polygonal regions and lines allowing convenient and efficient cell data access. In post-processing, we initialize a grid map instance with a range setting of \([6, 46]\) meters in x-coordinate and \([-10, 10]\) meters in y-coordinate. The resolution is set to 0.05 meter per cell so that the grid map has 800 cells in x-coordinate and 400 cells in y-coordinate. When the post-processing node receives a network output frame, it stores the frame as an Eigen matrix. In Algorithm 2, Step 1-3 is processed on the matrix. In Step 4-5, the contour vertices are imported to a polygon iterator instance and then the drivable region is labeled cell by cell as the polygon iterator iterates. The execution time of post-processing is 5ms per frame.

For visualization purpose, we also generate a drivable region map on camera view by applying a similar post-processing procedure.

Algorithm 2 Post-processing of CNN output as segmentation results

Data Input tensor \( I=\{I_{i,j}\} \), output tensor \( P=\{P_{i,j}\} \)

Parameter threshold \( \text{THR} \).

1: \( I \leftarrow \text{Threshold}(I \geq \text{THR}) \)
2: \( I \leftarrow \text{GetLargestConnectedComponent}(I) \)
3: \( I \leftarrow \text{Dilation}(I, \text{disk}(1)) \)
4: \( B \leftarrow \text{GetContour}(I) \)
5: \( A \leftarrow \text{Polygon}(B) \)
6: return \( A \)

D. Network training and evaluation

The training platform of ChipNet is on a workstation with an Xeon 2.4 GHz CPU and an NVidia K20c GPU. The software environment is a Python based framework named Keras with Tensorflow 1.4 back end. The input of the network is an \(180 \times 64 \times 14\) tensor and the output of the network is an \(180 \times 64 \times 1\) tensor. The training speed on the platform is 256 ms per frame. To evaluate the performance of the proposed solution, a subset of the Ford Campus Vision and LiDAR Dataset [27] and the KITTI road benchmark [11] is used for training and testing purposes.

The original Ford Dataset described in [27] contains 3871 frames of LiDAR data recorded with synchronized camera data. The LiDAR data are sampled at 10 Hz. The dataset itself had no labels or annotations, so we created a subset and labeled the drivable region manually. To reduce the overlaps among the consecutive frames, we selected only 1 out of every 5 consecutive frames. Effectively the dataset is downsampled to 2 frames per second. We also removed some off-road samples, such as vehicles on the parking lot, from the dataset since we concentrated on road scenarios. Therefore, we generated a 600-frame subset from the Ford dataset for training and evaluation.

In the subset, the original image is cropped from the size of \(1243 \times 1616\) to \(800 \times 200\) resolution that overlaps with the LiDAR point cloud. The data is arranged as described in Section III-A. In order to obtain the LiDAR ground truth, a ray tracing approach as described in Algorithm 4 is applied. The projection method from LiDAR coordinate to camera coordinate is described in Algorithm 5. In our labeled subset, each sample includes a \(180 \times 64 \times 14\) LiDAR frame, a \(800 \times 200\) color image, a \(180 \times 64 \times 1\) LiDAR ground truth frame and a \(800 \times 200\) ground truth image. We randomly selected 400 samples for training and validation and the remaining 200 samples for testing.

Furthermore, we augmented the training samples through rotating the field of view by \((-10, -5, 0, 5, 10)\) degrees from the LiDAR ground truth. Thus, we generated a training set with 2000 samples. Cross entropy was selected as the loss function and Adam [21] method with default settings was selected as the optimizer. We first trained the network without the quantization plug-in for 30 epochs, at which time the training process convergences well. We then fine-tuned the network with the quantization plug-in for 10 epochs to obtain the optimal fixed-point weights. The first training took 4.5 hours and each fine-turning took 1.5 hours. For each defined fixed-point bit-length format, we applied the same simulated quantization procedure during training. The bit length with the least loss is chosen for the FPGA implementation.

In the testing session, we selected F1 score (F1), average precision (AP), precision (PRE), recall (REC), false positive rate (FPR) and false negative rate (FNR) in image view as the evaluating metrics. The metrics are computed as in (1-4). Table 6 presents the evaluation results using different bit length of fixed-point quantization. The result shows that the proposed network quantized to 16 or more bits has comparable accuracy to floating-point results, but accuracy drops sharply
Algorithm 3: Ground truth labeling for LiDAR samples

\[ B = \{ B_{i,j} \} \]

1. \[ \vec{x}_1 \leftarrow \begin{bmatrix} L_{i,j,1} \\ L_{i,j,2} \\ L_{i,j,3} \\ L_{i,j,10} \end{bmatrix}, \quad \vec{x}_2 \leftarrow \begin{bmatrix} L_{i,j,8} \\ L_{i,j,9} \end{bmatrix} \]
2. \[ \vec{x}_1 \leftarrow \Proj(\vec{x}_1), \quad \vec{x}_2 \leftarrow \Proj(\vec{x}_2) \]
3. \[ G_{i,j,1} \leftarrow [B(\vec{x}_1) > 0] \times [B(\vec{x}_2) > 0] \]
4. return \( G \)

Algorithm 4: Projection from LiDAR coordinate to camera coordinate

\[ \text{Data LiDAR point} \ P_{x_i, y_i, z_i, r_i} \]
\[ \text{Parameter transform matrix} \ K \in R^{3 \times 4} \]

1. \[ \begin{bmatrix} \hat{x}_i \\ \hat{y}_i \\ \hat{z}_i \\ 1 \end{bmatrix} \leftarrow K \begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix} \]
2. \[ \begin{bmatrix} \hat{x}_i \\ \hat{y}_i \\ \hat{z}_i \end{bmatrix} \leftarrow \begin{bmatrix} \frac{x_i}{z_i} \\ \frac{y_i}{z_i} \end{bmatrix} \]
3. return \[ \begin{bmatrix} \hat{x}_i \\ \hat{y}_i \end{bmatrix} \]

if quantization is below 16 bits. In our work, 18 bits are selected since it is the best choice supported by the target FPGA platform.

\[
\text{Precision} = \frac{TP}{TP + FP} \quad (1)
\]

\[
\text{Recall} = \frac{TP}{TP + FN} \quad (2)
\]

\[
F1 \text{ score} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \quad (3)
\]

\[
AP = \frac{TP + TN}{TP + FP + TN + FN} \quad (4)
\]

We also evaluate our network in KITTI road benchmark [11]. The KITTI vision benchmark suite is a widely used dataset that contains LiDAR, camera, GPS and IMU data records. In addition, the vertex transformation from LiDAR coordinate to camera coordinate is provided. The road benchmark in the suite includes 289 training samples and 290 test samples. In the benchmark, the point cloud comes from a 64-line Velodyne laser scanner and the camera frame comes from a Point Grey 1.4 megapixels camera. For better sensor fusion, the LiDAR point cloud is rectified at each time step and the camera frame is cropped to a 375 × 1242 image. In addition, the data frames from different sensors are synchronized to 10 Hz.

Different from the Ford dataset that evaluates on camera view, the KITTI road benchmark evaluates the segmentation results on bird eye view, in which the result is mapped to a 400 × 800 image. The mapped image represents the accessibility of the region of [6, 46] meters in the front and 10 meters on both left and right sides.

In the training session, we first augmented the dataset through rotating the field of view by \((-10, -9, -8, -7, -6, -5, -4, -3, 0, 2, 4, 6, 8, 10)\) degrees from the corresponding LiDAR ground truth. So, we obtained 3179 samples, among that 3000 randomly selected samples are used for training and the other 179 samples are used for validation. We fine-tuned the network with quantization plug-in for 10 epochs from the weights trained in Ford dataset, and submitted the results to the benchmark online evaluator. The training time was 2.05 hours. A comparison with several existing results is presented in Table IV. Typical results are shown in Figure 7. The red area denotes the false drivable region (false positive), the blue area denotes the missing driving region (false negative), the green area denotes the correct drivable region (true positive), and the rest area denotes the correct forbidden region (true negative) or don’t care region.

Our proposed approach can provide highly reliable drivable region segmentation with minor distortions around the road boundary. For vehicles on the road, the segment boundary matches the ground truth boundary or slightly distorts towards the road center that is safe for automated driving. For the road with sidewalk, the segment boundary matches the ground truth if the sidewalk is above the road surface. However, if the sidewalk is equal or below than the road surface, the detected drivable region sometimes extends 1 to 2 meters into

![Figure 6](image)

Figure 6: Examples of the segmentation results from Ford dataset
the sidewalk, which needs to be improved in our future work. In addition, our solution returns accurate drivable regions in poor illumination scenarios such as inside tunnels or facing the sun glare. In contrast, vision based solutions rarely work well in those scenarios.

IV. HARDWARE ARCHITECTURE

As described in Section III, the LiDAR data after pre-processing has 14 channels and the input data size is $180 \times 64$. After the first layer of convolutional encoding, it becomes a feature map with 64 channels. In the next 10 convolution layers, the input and output feature map sizes remain the same as $180 \times 64 \times 64$. The final layer performs the channel-wise mapping that produces an output map of $180 \times 64$, each indicating the possibility of drivable regions. The block diagram of hardware architecture is illustrated in Figure 8. The system consists of a 3D convolution unit, a ReLU block, a feature map buffer and an intermediate buffer. 2D convolution and adder trees are embedded in the 3D convolution block. Since the feature maps in each stage of ChipNet have the same size, this 3D convolution unit is used repetitively. A finite state machine (FSM) is designed to control the iterative processing steps.

A. Zero padding

In order to properly process the information along the boundaries, zero padding must be applied to the feature map produced by the convolution layer output. In our system, a dual-port RAM is implemented for automatic zero-padding. In Figure 9, all memory locations are pre-loaded with zeroes. Pixels of a feature map are written to the corresponding address locations in the feature map buffer. When reading the feature map from the feature map buffer in continuous addresses, data are automatically zero-padded. The RAM functions as the feature map buffer.

B. Convolution

As exhibited in Figure 10, the 3D convolution unit contains 64 pieces of 2D convolution slices. Each 2D convolution slice is built with a line buffer and two $5 \times 5$ multiplier arrays. The line buffer is designed using shift registers as shown in Figure 11. It outputs a $5 \times 5$ window (outlined in red) as the input to the multiplier arrays. The registers in green multiply with the dilated $3 \times 3$ convolution kernel, and registers in yellow multiply with the regular $3 \times 3$ convolution kernel, and the register at the center multiplies with the coefficient sum as shown in Figure 4.

In each 2D convolution block, the input data are fed from the line buffer to two multiplier arrays, each followed by an adder tree. The 2D convolution block is a pipeline architecture that can process two convolution kernel operations in parallel. Since each 2D convolution operation has 64 convolution kernels, the same feature map is reloaded and processed for 32 times. All weights are stored in on-chip memory to avoid the latency of off-chip memory access. The ReLU block is implemented by a comparator and a multiplexer. If the input value is larger than 0, it outputs the original value. Otherwise the ReLU block outputs 0.

C. FSM Controller

Since the multiplier array in Figure 10 consumes a large number of DSP slices on FPGA, reusing it for each convolu-
Table IV: Comparison with existing results on KITTI road benchmark

| Name                          | F1 score % | AP %  | PRE % | REC % | FPR % | FNR % | Runtime ms |
|-------------------------------|------------|-------|-------|-------|-------|-------|------------|
| ChipNet on FPGA (this work)   | 94.05      | 88.29 | 93.57 | 94.53 | 3.58  | 5.47  | 17.59      |
| LoDNN [3]                     | 94.07      | 92.03 | 92.81 | 95.37 | 4.07  | 4.63  | 18         |
| HybridCRF [38]               | 90.81      | 86.01 | 91.05 | 90.57 | 4.90  | 9.43  | 1500       |
| LidarHisto [4]               | 90.67      | 84.79 | 93.06 | 88.41 | 3.63  | 11.59 | 100        |
| MixedCRF [16]                | 90.59      | 84.24 | 89.14 | 92.13 | 6.20  | 7.87  | 6000       |
| FusedCRF [37]                | 88.25      | 79.23 | 83.62 | 93.44 | 10.08 | 6.56  | 2000       |
| RES3D-Velo [34]              | 86.58      | 78.34 | 82.63 | 90.92 | 10.33 | 9.08  | 60         |

D. Implementation Results

The target hardware platform is Xilinx UltraScale XCKU115 FPGA. An integrated test system is demonstrated in Fig 12. The point cloud data is transmitted into PC via UDP protocol. After pre-processing, 14 channels input feature maps are sent into FPGA through ethernet interface. System clock frequency is set to 350 MHz. Each convolution block takes about 12,512 clock cycles to generate 2 feature maps. The total processing time of this CNN architecture is about 12.59 ms. Since normally LiDAR point cloud data frame rate is 10 Hz, this FPGA implementation fulfills the requirement of real-time LiDAR data processing. When running ChipNet in software on the Intel Core i5-5200U CPU, the processing time is 549 ms; when running ChipNet using the Nvida K20 GPU, the processing time is 162 ms. Thus, the FPGA implementation gains 43x speed up over CPU and 13x speed up over GPU. As mentioned earlier, there are few FPGA implementations of LiDAR processing using CNN at this time, performance and efficiency comparison with similar works on FPGAs is not available. Concerning to the power consumption, the design costs 37.5W. This is only 35% when compared to that of GPU, which is 107W.

The resource usage of our proposed neural network is listed in Table V.

Table V: Resource usage on the FPGA implementation of ChipNet

| FPGA Resource | Used  | Available | Utilization |
|---------------|-------|-----------|-------------|
| Slice Registers | 33530 | 1226720   | 2.73%       |
| Slice LUTs     | 38082 | 663360    | 5.74%       |
| Block RAMs     | 1543  | 2160      | 71.44%      |
| DSPs           | 3072  | 5520      | 55.65%      |

V. Conclusions

In this paper, the problem of drivable region segmentation is framed as a semantic segmentation task by processing real-time LiDAR data using a convolutional neural network on an FPGA. The LiDAR data is organized and sampled to a dense input tensor during pre-processing. An efficient and extendable CNN architecture namely ChipNet is proposed as the main processor. A reusable and efficient 3D convolution block is designed for FPGA implementation. The proposed approach is trained using Ford dataset as well as the KITTI benchmarks. Evaluation shows the proposed LiDAR processing algorithm can achieve state-of-art performance in accuracy and also real-time processing in speed. However, the FPGA implementation still consumes a large amount of on-chip memory. For future work, we will consider recurrent neural network for spatial-sequence decoding that may reduce the on-chip memory usage. We also notice during benchmark evaluation that sidewalk and railway are the main causes of false positives. Sensor fusion of
LiDAR and camera data will be considered to further improve the accuracy. In addition, we plan to implement the proposed solution on our autonomous vehicle prototype for field test.

REFERENCES

[1] José M Álvarez Alvarez and Antonio M Lopez. Road detection based on illuminant invariance. IEEE Transactions on Intelligent Transportation Systems, 12(1):184–193, 2011.

[2] 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.

[3] Luca Caltagirone, Samuel Scheidegger, Lennart Svensson, and Mattias Wahde. Fast lidar-based road detection using fully convolutional neural networks. In Intelligent Vehicles Symposium (IV), 2017 IEEE, pages 1019–1024. IEEE, 2017.

[4] Liang Chen, Jian Yang, and Hui Kong. Lidar-histogram for fast road and obstacle detection. In Robotics and Automation (ICRA), 2017 IEEE International Conference on, pages 1343–1348. IEEE, 2017.

[5] Xiaozhi Chen, Kaustubh Kundu, Yukun Zhu, Andrew G Bermeshawi, Huimin Ma, Sanja Fidler, and Raquel Urtasun. 3d object proposals for accurate object class detection. In Advances in Neural Information Processing Systems, pages 424–432, 2015.

[6] Xiaozhi Chen, Huimin Ma, Ji Wan, Bo Li, and Tian Xia. Multi-view 3d object detection network for autonomous driving. In IEEE CVPR, volume 1, page 3, 2017.

[7] Zhe Chen and Zijin Chen. Rnnet: A deep neural network for unified road and road boundary detection. In International Conference on Neural Information Processing, pages 667–677. Springer, 2017.

[8] A. Cosgun, L. Ma, J. Chiu, J. Huang, M. Demir, A. M. Aifon, T. Lian, H. Tafish, and S. Al-Stouhi. Towards full automated drive in urban environments: A demonstration in gomuentum station, california. In 2017 IEEE Intelligent Vehicles Symposium (IV), pages 1811–1818, June 2017.

[9] Xingping Dong, Jianbing Shen, Ling Shao, and Luc Van Gool. Submarkov random walk for image segmentation. IEEE Transactions on Image Processing, 25(2):516–527, 2016.

[10] Peter Fankhauser and Marco Hutter. A universal grid map library: Implementation and use case for rough terrain navigation. chapter 5. Springer, 2016.

[11] Jannik Fritsch, Tobias Kuehnel, and Andreas Geiger. A new performance measure and evaluation benchmark for road detection algorithms. In International Conference on Intelligent Transportation Systems (ITSC), 2013.

[12] Noa Garnett, Shai Silberstein, Shaul Oron, Ethan Fetaya, Uri Verner, Peter Fankhauser and Marco Hutter. A universal grid map library: Implementation and use case for rough terrain navigation. chapter 5. Springer, 2016.

[13] Alejandro González, Gabriel Villalonga, Jialong Xu, David Vázquez, Jaume Amores, and Antonio M López. Multiview random forest of local experts combining rgb and lidar data for pedestrian detection. In Intelligent Vehicles Symposium (IV), 2015 IEEE, pages 356–361. IEEE, 2015.

[14] S. Gu, Y. Zhang, J. Yang, and H. Kong. Lidar-based urban road detection by histograms of normalized inverse depths and line scanning. In 2017 European Conference on Mobile Robots (ECMR), pages 1–6, Sept 2017.

[15] Xiaofeng Han, Huan Wang, Jianfeng Lu, and Chunxia Zhao. Road detection based on the fusion of lidar and image data. International Journal of Advanced Robotic Systems, 14(6):17299881417738102, 2017.

[16] Xiaofeng Han, Huan Wang, Jianfeng Lu, and Chunxia Zhao. Road detection based on the fusion of lidar and image data. 14(6):17299881417738102, 2017.

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 770–778, 2016.

[18] 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.

[19] Brody Huval, Tao Wang, Sameep Tandon, Jeff Kiske, Will Song, Joel Pahayampallil, Mykhaylo Andriluka, Pranav Rajpurkar, Toki Migi-matsu, Royce Cheng-Yue, et al. An empirical evaluation of deep learning on highway driving. arXiv preprint arXiv:1504.01716, 2015.

[20] Benoit Jacob, Skirantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalinenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. arXiv preprint arXiv:1712.05877, 2017.

[21] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

[22] J. Levinson, J. Askeland, J. Becker, J. Dolson, D. Held, S. Kammel, J. Z. Kolter, D. Langer, O. Pink, V. Pratt, M. Sokolsky, G. Stanek, D. Stavens, A. Teichman, M. Welting, and S. Thrun. Towards fully autonomous driving: Systems and algorithms. In 2011 IEEE Intelligent Vehicles Symposium (IV), pages 163–168, June 2011.

[23] Bo Li. 3d fully convolutional network for vehicle detection in point cloud. In Intelligent Robots and Systems (IROS 2016), 2017 IEEE/RSS International Conference on, pages 1513–1518. IEEE, 2017.

[24] Xiaolong Liu and Zhidong Deng. A graph-based nonparametric drivable road region segmentation approach for driverless car based on lidar data. In Proceedings of the 2015 Chinese Intelligent Automation Conference, pages 431–439. Springer, 2015.

[25] 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.

[26] Gabriel Leivas Oliveira, Wolfram Burgard, and Thomas Brox. Efficient deep methods for monocular road segmentation. In IEEE/RSS international conference on intelligent robots and systems (IROS 2016), 2016.

[27] Gaurav Pandey, James R McBride, and Ryan M Eustice. Ford campus driving: Systems and algorithms. In Extreme Computing Conference (HPEC), 2014 IEEE, pages 1–6, Sept 2017.

[28] Jianteng Peng, Jianbing Shen, and Xuelong Li. High-order energies for stereo segmentation. IEEE transactions on cybernetics, 46(7):1616–1627, 2016.

[29] Jianting Shen, Yunfan Du, Xuelong Li, et al. Interactive segmentation using constrained laplacian optimization. IEEE Trans. Circuits Syst. Video Techn., 24(7):1088–1100, 2014.

[30] Jianting Shen, Yunfan Du, Wenguang Wang, and Xuelong Li. Lazy random walks for superpixel segmentation. IEEE Transactions on Image Processing, 23(4):1451–1462, 2014.

[31] Jianting Shen, Xiaoqing Liu, Zhiyuan Jiang, Yu Liu, Wenguang Wang, and Ling Shao. Real-time superpixel segmentation by dbscan clustering algorithm. IEEE Transactions on Image Processing, 25(12):5933–5942, 2016.

[32] Jianting Shen, Jianting Peng, Xingping Dong, Ling Shao, and Fatih Porikli. Higher order energies for image segmentation. IEEE Transactions on Image Processing, 26(10):4911–4922, 2017.

[33] Patrick Y Shinzato, Diego Gomes, and Denis F Wolf. Road estimation with sparse 3d points from stereo data. In Intelligent Transportation Systems (ITSC), 2014 IEEE 17th International Conference on, pages 1688–1693. IEEE, 2014.

[34] Patrick Yuri Shinzato, Denis Fernando Wolf, and Christoph Stiller. Road terrain detection: Avoiding common obstacle detection assumptions using sensor fusion. In Intelligent Vehicles Symposium (IV), 2014.

[35] Nicolas Soquet, Didier Aubert, and Nicolas Hautiere. Road segmentation supervised by an extended v-disparity algorithm for autonomous navigation. In Intelligent Vehicles Symposium, 2007 IEEE, pages 160–165. IEEE, 2007.

[36] Wei Wang and Xinming Huang. An fpga co-processor for adaptive lane detection based on the fusion of lidar and image data. In Proceedings of the IEEE International Conference on Intelligent Transportation Systems (ITSC), 2016.

[37] Yin Zhou and Oncel Tuzel. Voxelnet: End-to-end learning for point cloud based 3d object detection. CoRR, abs/1711.06396, 2017.
Yecheng Lyu (S’17) received his B.S. degree from Wuhan University, China in 2012 and M.S. degree from Worcester Polytechnic Institute, USA in 2015 where he is currently a Ph.D student working on autonomous vehicles. His current research interest is sensor fusion, autonomous vehicle perception and deep learning.

Lin Bai (S’18) received the B.S degree in integrated circuits design and integrated system from University of Electronic Science and Technology of China in 2009 and M.S. degree in electrical engineering and information technology from Swiss Federal Institute of Technology Zurich in 2012. Previously he was an FPGA engineer in industry. He is currently working towards the Ph.D. degree at Worcester Polytechnic Institute, USA. His current research interest is hardware acceleration of deep learning algorithms on FPGA and ASIC.

Xinming Huang (M’01–SM’09) received the Ph.D. degree in electrical engineering from Virginia Tech in 2001. Since 2006, he has been a faculty in the Department of Electrical and Computer Engineering at Worcester Polytechnic Institute (WPI), where he is currently a chair professor. Previously he was a Member of Technical Staffs with the Bell Labs of Lucent Technologies. His main research interests are in the areas of circuits and systems, with emphasis on autonomous vehicles, deep learning, IoT and wireless communications.