Evaluation of Quantized LaneNet on Closed-loop System

An, Mingyu

Award date:
2021

Department of Electrical Engineering
Electronic Systems Research Group

Evaluation of Quantized LaneNet on Closed-loop System

Master Thesis Report

Mingyu An
1363921

Supervisors:
Assistant Professor Dr. Dip Goswami
Ph.D. Candidate Sayandip De
Ph.D. Candidate Sajid Mohamed

4.0 version

Eindhoven, July 2021

Contents

Contents iii

List of Figures v

List of Tables vi

1 Introduction 1
- 1.1 Introduction . . . 1
- 1.2 Motivation . . . 2
- 1.3 Report Structure . . . 5

2 Related Work 6
- 2.1 ADAS . . . 6
- 2.2 Lane Detection . . . 6
  - 2.2.1 Traditional lane detection algorithm . . . 6
  - 2.2.2 Lane detection algorithm with deep learning . . . 7
- 2.3 Quantization . . . 8

3 Problem Statement 10
- 3.1 Problem Statement . . . 10
- 3.2 Research Questions . . . 10

4 LaneNet 11
- 4.1 LaneNet . . . 11
  - 4.1.1 Binary Segmentation . . . 11
  - 4.1.2 Instance Segmentation . . . 12
  - 4.1.3 Clustering . . . 12

5 LaneNet Quantization 13
- 5.1 Background . . . 13
  - 5.1.1 Fixed-point Number Format . . . 13
  - 5.1.2 Constraint . . . 14
- 5.2 Quantization Procedure . . . 15

6 Experimental Setup 18
- 6.1 Experimental Setup . . . 18
- 6.2 Dataset . . . 19
  - 6.2.1 TuSimple . . . 19
  - 6.2.2 Custom dataset . . . 19
  - 6.2.3 Evaluation . . . 21

7 Results and Discussions **22**
7.1 Quantization of the LaneNet with TuSimple dataset 22
7.2 Quantization of the LaneNet with custom dataset 23
7.3 The comparison of the results of two datasets 25

8 Conclusions and Future Works **26**
8.1 Conclusion 26
8.2 Future work 26

Bibliography **27**

List of Figures

1.1 Examples of ADAS, image source: [1], [2], [3] 1
1.2 General lane detection system 2
1.3 An example of CNN architecture (image source: [4]) 2
1.4 Convolution layer structure 3
1.5 Examples of some activation functions 3
1.6 Example of max pooling and average pooling 4
1.7 Example of fully-connected layer 4

2.1 Comparison of methods to reduce precision on AlexNet, image source: [5] 9

4.1 Lanenet architecture 11
4.2 The intra-cluster pulling force pulls embeddings towards the cluster center, i.e. the mean embedding of that cluster. The inter-cluster repelling force pushes cluster centers away from each other. 12

5.1 Example of fixed-point representation 14
5.2 Example of fixed-point multiply-accumulation 14
5.3 Quantization procedure 15
5.4 Simulate fixed-point convolution layer during forward pass 16

6.1 Structure of LaneNet 18
6.2 Examples of TuSimple 19
6.3 Examples of different lane types in custom dataset 20
6.4 Examples of different weathers in custom dataset 20
6.5 Examples of different road layout in custom dataset 21

7.1 The output of the trained LaneNet with test TuSimple dataset 22
7.2 The Pareto diagram of only quantized weight and only quantized input with TuSimple dataset 23
7.3 The output of the trained LaneNet with test custom dataset 24
7.4 The Pareto diagram of only quantized weight and only quantized input with custom dataset 24

List of Tables

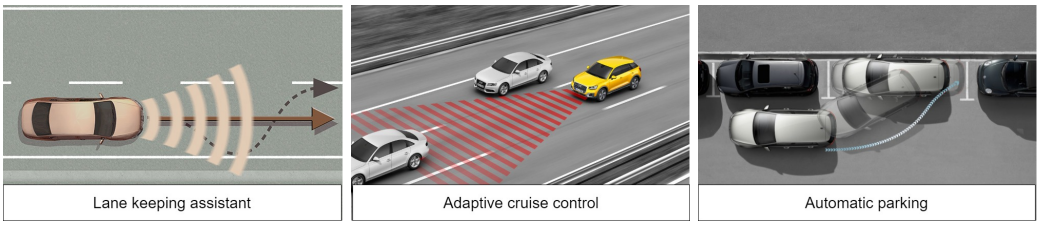
7.1 The result of iteratively quantizing weight and input data with TuSimple dataset . 22
7.2 The result of iteratively quantizing weight and input data with custom dataset . . 23

Chapter 1

Introduction

1.1 Introduction

Autonomous driving technology has the potential to reduce crashes, prevent injuries, and save lives. It has been a challenging field of research since the middle 1980s. According to the report from National Highway Traffic Safety Administration (NHTSA), approximately 94% of all serious motor vehicle crashes are due to human error or choices [6], for example, distraction, emotional driving, drowsy driving. Although fully autonomous vehicles are still in the research stage, the autonomous vehicle manufacturers have already developed systems which act as assistance features on a vehicle, i.e. advanced driver-assistance systems (ADAS). Many autonomous driving assistance technologies have already been applied to vehicles in market, such as lane keeping assist, adaptive cruise control, automatic parking, etc. Among these systems, the perception system which is similar to the human vision, through the use of sensors and controllers to let the vehicle fully understanding the surrounding environments. In this case, we focus on the camera-based lane detection makes an important contribution to such environment perception.



(a) (b) (c)

Figure 1.1: Examples of ADAS, image source: [1], [2], [3]

Camera-based lane detection [7, 8, 9, 10, 11, 12] uses the image data from the camera to enable the vehicle to recognize the lane markers, which could further allow the vehicle to properly position itself within the lanes. Figure 1.2 shows a general process of lane detection system [13], including image pre-processing, feature extraction and model fitting. In the image pre-processing stage, there are several operations, such as region of interest (ROI) selection, transferring color image into grayscale image or a different color format, noise removal and blur, etc, which can be applied to the raw image obtained from the camera to reduce clutter and enhance features of interest. After pre-processing, lane features, such as colors and edge features can be extracted and hence, can be detected based on these features. Traditional lane detection methods mostly deal with simple road scenes which are prone to robustness issues due to the variation of the environment, such as illumination variation, target occlusion, blurred road edges, etc. In addition,

these highly specialized, handcrafted features fail to express the semantic features of the objects in new scenes, resulting in insufficient detection accuracy that can hardly meet the safety requirements of autonomous vehicles. With the breakthroughs in deep learning, recent methods replace the handcrafted feature detectors with deep neural network to learn pixel-wise lane segmentation prediction through big data, allowing it to better adapt to complex environments and to gain a better accuracy.

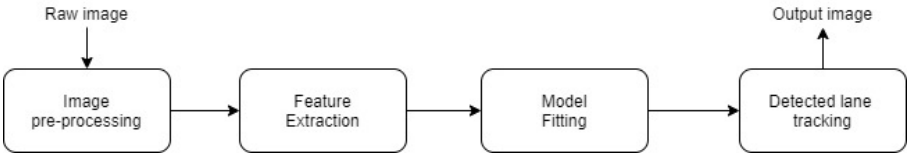


Figure 1.2: General lane detection system

Lane detection algorithm, as a data-driven computer vision algorithm, is extremely computationally intensive [14]. It needs to extract features from an abundance of images, train models with tremendous data samples and deploy those models on query images. [15] proves that a significant redundancy in the parameterization of many CNNs. With less weight values for each feature, the model can still accurately predict the remaining values. They trained several different architectures by learning only a small number of weights and predicting the rest. In the best of circumstances, they were able to predict more than 95% of the weights of a network without any drop in accuracy. Therefore, in order to get a more efficient model, compressing neural network is very essential. Quantization is one of the effective methods to compress model. Quantization in general is a method that approximates a neural network, by converting floating-point numbers to low bit-width numbers. This could extremely decrease the size of the model, whereas it will lose a certain degree of accuracy.

1.2 Motivation

Lane detection algorithm has already been developed using deep learning techniques in recent efforts. Various lane detection models have been proposed with high accuracies, such as PINet [16], ENet-SAD [17], LaneNet [18] etc. This thesis will mainly focus on a model using convolutional neural network (CNN). As shown in Figure 1.3 below, a typical CNN contains four main layers, convolution layers, activation layers, pooling layers, and fully-connected layers.

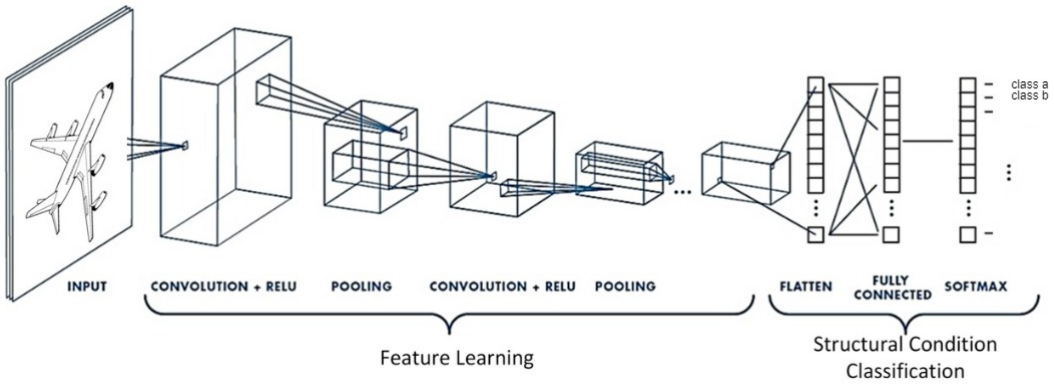


Figure 1.3: An example of CNN architecture (image source: [4])

The convolution layer is the core building block of a CNN, which is the first layer to extract features from the input image. When the input image is passed through a convolution layer, the features are extracted by applying a 2D spatial convolution to the image. As Figure 1.4 illustrated, filtering

is the primary step in this process. There is a two-dimensional array of weights, representing part of the image, called kernel. It will move across the receptive fields of the image, checking whether the feature is present or not. This means when the kernel is applied to an area of the image, a dot product is calculated between the input pixels and the kernel, which will then be fed into an output array. Afterwards, the kernel will shift by stride S, repeating the above step until it has swept across the whole image. The final output array will be the output feature map.

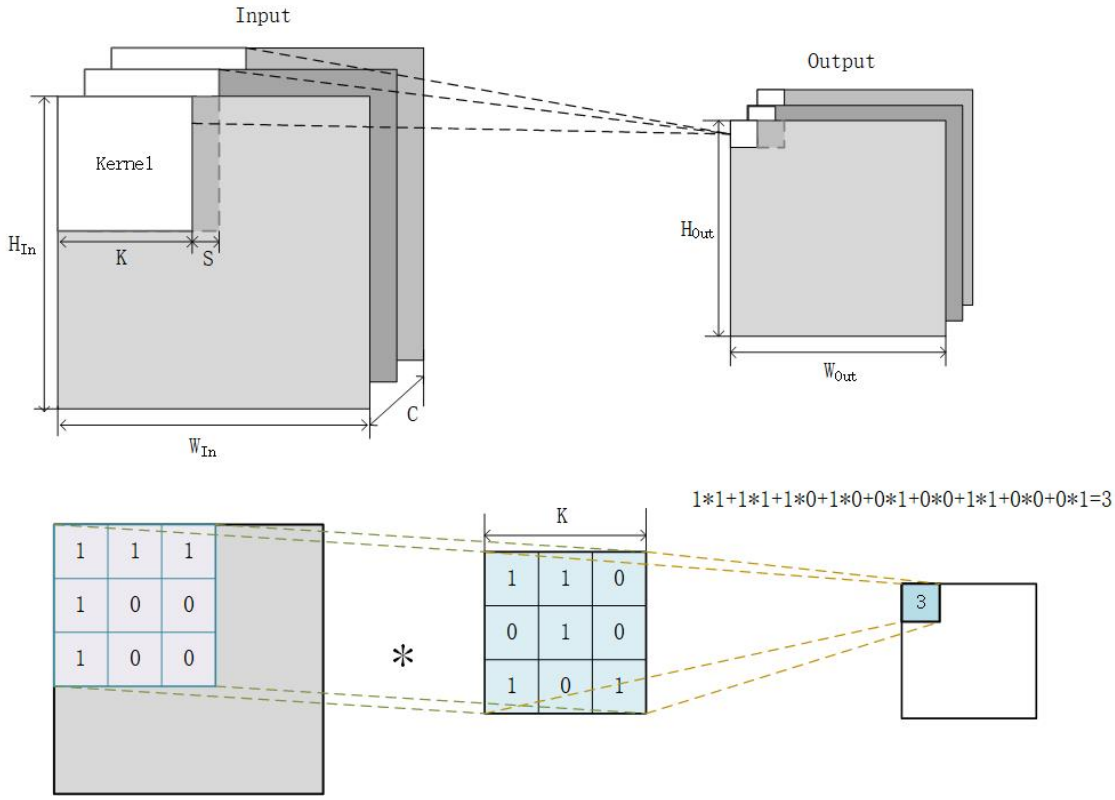
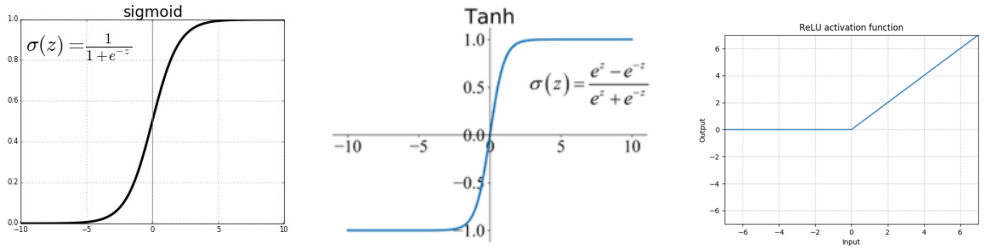


Figure 1.4: Convolution layer structure

Since convolution is a linear operator, the activation layer is very essential between layers, adding the non-linearities to the model. This could enable the network to learn complex functional mappings between the inputs and response variables during the training stage. There are many activation functions, such as Sigmoid, tanh, Rectified Linear Unit (ReLU), etc. Among them, ReLU ($ReLU(x) = max(0, x)$) is the most common used activate function in most of the CNN.



(a) (b) (c)

Figure 1.5: Examples of some activation functions

The pooling layer may follow by the convolution layer after the activation function. It is responsible

for reducing the spatial size of the feature map as well as minimizing the possibility of over-fitting. The most common pooling methods are max pooling and average pooling. Max pooling will select the maximum value of a set of pixels within 2D windows, while average pooling will take the average value. An example is shown in Figure 1.6.

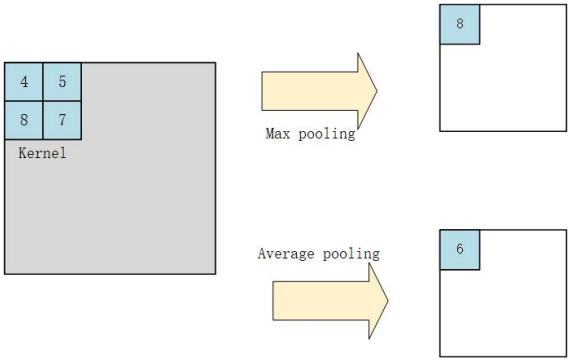


Figure 1.6: Example of max pooling and average pooling

The last few layers in CNN are fully-connected layers or classification. The layer combines all the features extracted through the previous layers and classifies the input image. The layer structure is as shown in Figure 1.7. Each node in the output layer will directly connect to the node in the previous layer. Since the fully-connected layer is very memory-intensive, if possible, it should be replaced by the convolution layer [19].

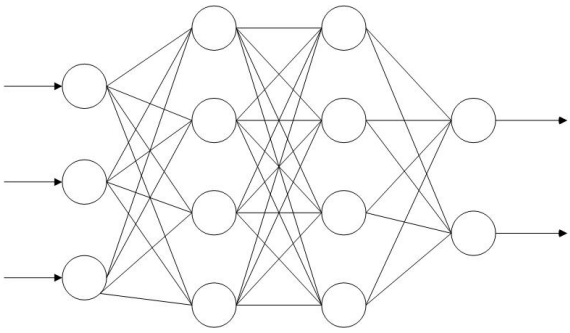


Figure 1.7: Example of fully-connected layer

Normally in a convolutional neural network (CNN), the convolutional layer is the most compute-heavy which particularly requires the computation of large dot products between the network parameters, i.e. the kernel weights and the feature map elements [20]. Assuming a convolutional kernel with a size of K and a feature map depth of C channels, these dot products comprise $K^2 \cdot C$ numeric multiplications for each application of a kernel over the width and height dimensions of the input feature map. This process is illustrated in Figure 1.4. This will be further duplicated, using the C' channels of the output feature map as the new input with the corresponding set of kernel parameters. The challenges for a CNN inference engine are the enormous dot-product compute. The challenge can be overcome by quantization.

Quantization is a method which changes the floating-point to fixed-point arithmetic eliminating the unnecessary precision from the network parameters. It can therefore reduce the model size and get a more efficient model. Even though the network will lose a certain degree of accuracy after quantization, while using the certain method with fine-tuning, the accuracy loss could be very low. Thus, quantization is an extremely effective method to make the model more efficient.

1.3 Report Structure

The organization of this thesis is as follows: Chapter 1 introduces the project. Chapter 2 discusses the related works and problems and contributions are listed in Chapter 3. Chapter 4 and 5 introduce the background information of LaneNet and the quantization of LaneNet. Chapter 6 introduces the experimental set up and the datasets. Chapter 7 shows the result of quantized LaneNet with different datasets. Chapter 8 gives the conclusion and discusses the future works.

Chapter 2

Related Work

2.1 ADAS

The LKAS, as the most relevant ADAS to lane detection algorithm, is a further development of the modern lane-departure warning system (LDWS) that has the ability to support the driver in staying within a lane. The first production LDWS in Europe was developed by the United States company Iteris for Mercedes Actros commercial trucks [21] in 2000 and it is still available on many new cars, SUV's and trucks nowadays. In 2003, Honda launches its LKAS on the Inspire [22], which provided up to 80% of steering torque to keep the vehicle in its lane on the highway [23]. In the system, a camera mounted inside the upper front windshield applying the appropriate input and the control unit figured out the appropriate assist steering torque and further control the electric power steering (EPS) of vehicle. In 2006, Lexus presented a multi-mode LKAS on the LS 460, which used stereo cameras and more sophisticated object- and pattern-recognition processors [24]. In 2007, Audi began offering its Audi Lane Assist feature on the Q7 [25], which will vibrate the steering wheel if the vehicle appears to be exiting its lane. Recently, Tesla Model S [26] presented a advanced lane assistance system combining with a speed assist feature where the front facing camera uses the technology of the computer vision character recognition system to read the traffic speed limit and then transmit it to the car. While the traffic signs are absent on road, the system will rely on the GPS data. When the car moves away from a lane at above 48 km/h, the system beeps and the steering wheel vibrate, alerting the driver of an unintended lane change. Volvo [27] presented a Pilot Assistant working in conjunction with Adaptive Cruise Control to provide steering assistance to help keep the vehicle in its lane at a set speed and distance (or time) interval as long as there are clear markings on the road.

In 2001, Nissan Motors presented a lane keeping support system which targeted at 'monotonous driving' situations (when the drivers feel tired after long hours of continuous expressway driving) [28]. The system used a single CCD camera to estimate the road geometry and the host vehicle's position in the lane, a steering actuator to steer the front wheels and an electronic control unit to calculate the steering torque needed to keep within the lane.

2.2 Lane Detection

2.2.1 Traditional lane detection algorithm

The traditional approaches can basically be classified into two categories, model-based [29] and feature-based [30]. The model-based method believes that the lane can be represented by either the straight line or the parabolic curve, so that the way to detect the lane can be transform to calculate the model parameters. The feature-based method detects the lane by combining the low-level features, such as lane edges. Both of these methods rely on highly-specialized and hand-

crafted features likes color-based features [31] or shape-based features [32], and possibly combined with detection and tracking techniques such as Hough transform [33] or Kalman filter [34]. Tran et al. [31] proposed a lane marking detection method based on HSI color model. The method changed the way that the HSI color model calculated intensity component, making the intensity of pixels decrease. Hence, it became easier to apply the proper threshold to detect lane marking of the road. Kluge et al. [32] presented the LOIS lane detection algorithm. It used a deformable template approaches which formulated the lane detection problem in the form of finding the set of lane edge parameters. Then it applied the Metropolis algorithm to maximize the likelihood function which evaluated the quality of a given set of shape parameters matching the data in the road image. Therefore, it could determine the best set of lane shape parameters for the given image. Mariut et al.[32] introduced an algorithm that is able to detect lane marks, lane mark's characteristics and to determine the trend of vehicle's traveling direction. The algorithm first used a technique to estimate the mid-lane manually. Then, a magnitude image, highlighting the inner margins of the lane marks, was processed based on the mid-lane estimation. This could ensure the correctness of the detection by Hough transform. Borkar et al. [34] presented a night-time lane detection system integrated with pixel remapping, outlier removal, and prediction with tracking. The image would first be transformed to a bird's-eye view by inverse perspective mapping, and then be converted into a binary image by an adaptive threshold algorithm. After finding a set of lane marker candidates, the RANSAC algorithm was performed to eliminate the outlier. Finally, the method used the Kalman filter to track and smooth the lane estimation. Generally, these methods, however, are not robust to road scene variations, for instance, sudden illumination changes or weather conditions.

2.2.2 Lane detection algorithm with deep learning

With the successful development of deep learning, recent methods replace the hand-crafted feature detectors with a deep network to learn pixel-wise lane segmentation prediction. Gopalan et al.[35] proposed a learning based approach. It first used a pixel-hierarchy descriptor to model contextual information shared by lane markings and the surrounding region. Then it trained a classifier on the feature model using an outlier-robust boosting algorithms to learn relevant contextual features for detecting lane markings. Finally, it implemented the particle filtering framework to track the lane markings. A robust method based on the combined convolutional neural network (CNN) with random sample consensus (RANSAC) algorithm was introduced by Kim and Lee [36]. The method first applied a blurring and edge detection to removing the environment noises, as well as a hat-shape kernel to strengthen lane information. Then it detected the lane using the CNN combined with the RANSAC, in which the CNN works and provides the candidate of a road line after the failure of RANSAC. However, in this two methods, the CNN is only be used to detect lanes in complex real road with noisy environment, for instance, road side trees, fences or intersections. Huval et al.[37] presented their research on how existing CNNs can be used in vehicle and lane detection for real-time. They used a CNN based on Overfeat, converting the CNN into a "sliding window" detector. By providing a large resolution image, it could output an object mask, and by performing bounding box regression, it could predict a single bounding box for an object. He et al.[38] proposed a Dual-View CNN framework for lane detection. The method first extracted the lane line candidates by a designed weighted hat-like filter. Simultaneously, they utilized the front-view and top-view patches as the input to the DVCNN framework, in which the former eliminated the inferences caused by the surrounding environments like moving vehicles, barriers and curbs and the latter rejected non-club-shaped-structures such as ground arrows and words. Also a novel global optimization strategy which considered lane line probabilities, length, widths, orientations and the amount made the lane detection more accurate and robust. Li et al.[39] developed two kinds of neural networks to detect lanes in a real traffic scene. One is a CNN that simultaneously detects the lanes and the geometric attributes, i.e. location and orientation of the lane with respect to the region of interest. The other one is a recurrent neuron network (RNN) which can automatically detects lane boundaries. a multi-task CNN detecting the geometric lane attributes such as location and orientation and a recurrent neural network (RNN)

to infer the presence or absence of a lane over a sequence of image areas. Most recently, Neven et al.[18] proposed a branches, multi-task network, consisting of a lane segmentation branch and a lane embedding branch that can be trained end-to-end, casting the lane detection problem as an instance segmentation problem. Tabelini et al.[40] proposed PolyLaneNet, a CNN for end-to-end lane markings estimation.

2.3 Quantization

Deep learning has been proven to work well on tasks including image classification, Object detection, and natural language processing. With the increasing performance of the deep neural network (DNN) model, the depth of the neural network is getting deeper and deeper, which leads to the high storage and high power consumption of the model. This severely restricts the application of DNNs in resource-limited application environments and real-time online processing applications, especially intelligent mobile embedded devices, field programmable gate arrays (FPGA) and other online learning and recognition tasks. For example, the 8-layer AlexNet is equipped with 6,000 nodes and 6.1 million parameters, which requires 240MB of memory storage and 72.9 billion floating-point calculations (FLOPs) to classify a color image with a resolution of 224×224. At the same time, as the depth of the DNNs deepens, the storage cost will become greater. For the same image classification task mentioned above, using a VGGNet with 16 layers instead of the 8 layers AlexNet which has 150000 nodes and 144 million parameters, will cost 528MB memory storage and 15 billion floating-point calculation times. And using the ResNet-152 equipped with 557 million parameters requires 230 MB of memory storage and 11.3 billion floating-point calculation times. For mobile terminal equipment, FPGA and other weak storage and low computing power characteristics, it is impossible to directly store and run such a large deep network. On the one hand, large-scale DNN store a lot of redundant information which do not play a decisive role on the accuracy of the network. On the other hand, the performance of the simple DNNs cannot approach which of the large-scale deep neural networks. Therefore, compressing the original DNN and making it directly applied to the mobile embedded device will become an effective solution.

Quantization is one of the most commonly used model compressing approach. The main idea of quantization is replacing the original 32 bit floating parameters (full-precision parameters) by lower precision parameters. Gupta et al. [41] used 16 bit wide fixed-point arithmetic to present the full-precision float-point parameters and implemented the stochastic rounding scheme in training process, leading to a great improvement on the storage cost and computation performance. Using the dynamic fixed point quantization on AlexNet could almost achieve lossless compression. For instance, Ma et al. [42] quantized the weight and activation to 8 bit and 10 bit, respectively, without fine-tuning. Afterwards, Gysel et al. [43] quantized both the weight and activation to 8 bit with fine-tuning.

In order to reduce the storage cost and the number of float point computations in CNN to a greater extend, the idea that binarizing the network parameters has been widely proposed. BinaryConnect [44] reduced the convolution computations by directly quantizing the weight to -1 or 1. Whereas due to the network activation is still in full-precision, it couldn't substantially speeds up the network. Courbariaux et al. [45] presented BNN, which changed the original convolution calculation into Bitcount and XNOR. It greatly accelerated and compressed the model, while leading to the significantly reduction of the classification accuracy. For the purpose of reducing the precision loss, Rastegari et al. [46] proposed Binary-weight-network and XNOR-Networks, which implemented the scaling factor and kept the weight and input parameters of the first and last layer as 32-bit float point. It meanwhile reversed the order of convolution and regularization. Accompanied by these changes, BWN and XNOR-Net have achieved 0.8% and 11% increase in classification error rates compared to the original AlexNet, respectively. In recent works, by increasing the bit width of activation (greater than 1), and exploring combinations of different bit width of weights and activations, the performance of quantized network on ImageNet dataset is improved.

However, during training, there will be a gradient mismatch problem. Cai et al. [47] designed a half-wave Gaussian quantizer and proposed the HWGQ-Net, which effectively solved this problem.

Since the weights are approximately distributed in a Gaussian distribution with a mean value of 0, further considering 0 as the value after quantization may reduce the accuracy loss. Based on this, ternary weight net (TWN) [48] has been introduced. It constrained the weights to be ternary-valued: $+w$, 0 and $-w$. By changing the symmetrical w, trained ternary quantization (TTQ) [49] introduced different quantization factor, making the classification error rate on AlexNet only increase 0.6%. Figure 2.1 below shows the performance comparison of the above quantization network and the corresponding results of the quantized bit width.

Reduce precision Method		Bitwidth		$err_{top\text{-}5}$ ↑ /%
		Weights	Activations	
Dynamic Fixed Point	w/o fine-tuning[44]	8	10	0.4
	w/fine-tuning[45]	8	8	0.6
Reduce Weight	BC[46]	1	32(float)	19.2
	BWN[48]	1*	32(float)	0.8
	TWN[52]	2*	32(float)	3.7
	TTQ[53]	2*	32(float)	0.6
Reduce Weight and Activation	XNOR-Net[48]	1*	1*	11.0
	BNN[47]	1	1	29.8
	DoReFa-Net[49]	1*	2*	7.63
	QNN[50]	1	2*	6.5
	HWGQ-Net[51]	1*	2*	5.2

Notes: "*" denotes the method is not applied to first and/or last layers, and "↑" denotes increase.

Figure 2.1: Comparison of methods to reduce precision on AlexNet, image source: [5]

Chapter 3

Problem Statement

3.1 Problem Statement

Lane detection algorithms have already been developed by using CNNs. It has been found that the CNNs for large scale image recognition always face the issue of over-parameterized. The target to compress the network is the convolution layer, in which the computation of large dot products between the network parameters is required. Some researchers [50] have found that low-precision computation is sufficient for the network. Quantization, converting the floating-point parameters to a fixed-point representation is a potential solution to compress the network with a tolerable accuracy penalty.

The main purpose of this thesis is to quantize the lane detection neural network with TuSimple and custom datasets, respectively. Then the trade-off between the degree of quantization of CNN and the performance of the network will be discussed.

3.2 Research Questions

The main research questions of this thesis are:

- Train/validate/test the lane detection model LaneNet with TuSimple and custom datasets
- Quantize the floating-point lane detection model LaneNet to fixed-point and evaluate the performance.
- Analyse the trade-off between the degree of quantization of CNN and the performance model of CNN.

Chapter 4

LaneNet

This chapter introduces the background of LaneNet, the neural network used for lane detection in this project. The idea of this network comes from [18].

4.1 LaneNet

LaneNet is the neural network trained end-to-end for lane detection, in which the lane detection is treated as an instance segmentation problem. Hence, each lane forms its own instance with lane class. It can be split into two tasks, binary segmentation and instance segmentation. The binary segmentation task classifies the input pixels into two categories: the lane and the background. The instance segmentation further divides the segmented lane pixels into different lane instances. Then each lane pixel is assigned the id of their corresponding lanes. Therefore, the power of both tasks can be fully utilized without being distracted by other tasks, so that it eases the problem of lane changes and number of lanes can be handled. The architecture of this network is shown in Figure 4.1.

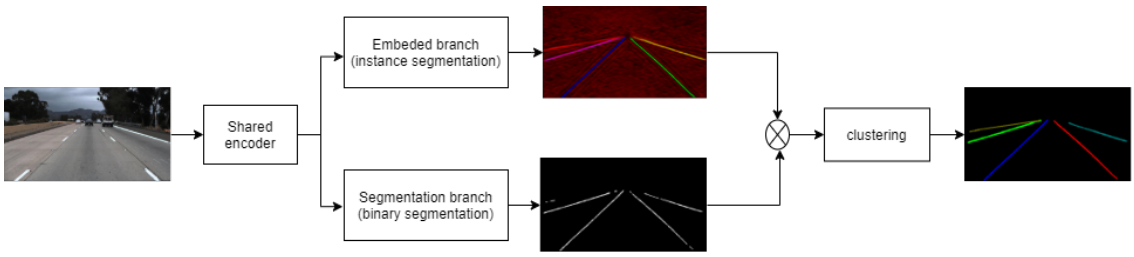


Figure 4.1: Lanenet architecture

4.1.1 Binary Segmentation

The binary segmentation branch is trained to output a binary segmentation map to indicate which pixel belongs to the lane and which do not. In the output segmentation map, the ground-truth lane points are connected together, which forms a connected line per lane. These lines cover through the objects like cars, and also the absence part of the visual lane segment, such as dashed lane and faded lane. By doing so, the network can learn to predict the location of the lane even when the circumstances are adverse. The binary segmentation network is trained with the standard cross-entropy loss function.

4.1.2 Instance Segmentation

The embedding branch is trained with discriminative loss function proposed by Bert De Brabandere el al [51] to output an embedding of each lane pixels, so that the distance between the embedding which belongs to the same lane is small, conversely, the distance between the embedding which belongs to the different lane is large. By doing this, the embeddings of the same lane will cluster together.

To form these unique clusters, per lane, two terms are introduced. The first one is a variance term (L_{var}) which represents a pull-force drawing each embeddings towards the mean embedding, i.e. the cluster center. The other one is a distance term (L_{dist}) which represents a push-force pushing cluster centers away from each other. Both terms are hinged: the pull-force is only active up when an embedding has a distance larger than δ_v from its cluster center and the push-force between centers is only active up when they are closer than δ_d to each other. This is illustrated in Figure 4.2. The total loss L can be written as follow:

$$\begin{cases} L_{var} = \frac{1}{C}\sum_{c=1}^{C}\frac{1}{N_c}\sum_{i=1}^{N_c}[\|\mu_c - x_i\| - \delta_v]_+^2 \\ L_{dist} = \frac{1}{C(C-1)}\sum_{c_A=1}^{C}\sum_{C_B=1,c_A\neq c_B}^{C}[\delta_d - \|\mu_{c_A} - \mu_{c_B}\|]_+^2 \\ L = L_{var} + L_{dist} \end{cases} \tag{4.1}$$

where C is the number of clusters, N_c is the number elements in cluster c, x_i is an embedding, μ_c is the cluster center, $\|\dots\|$ is the L2 distance, and $[x]_+ = max(0, x)$ is the hinge.

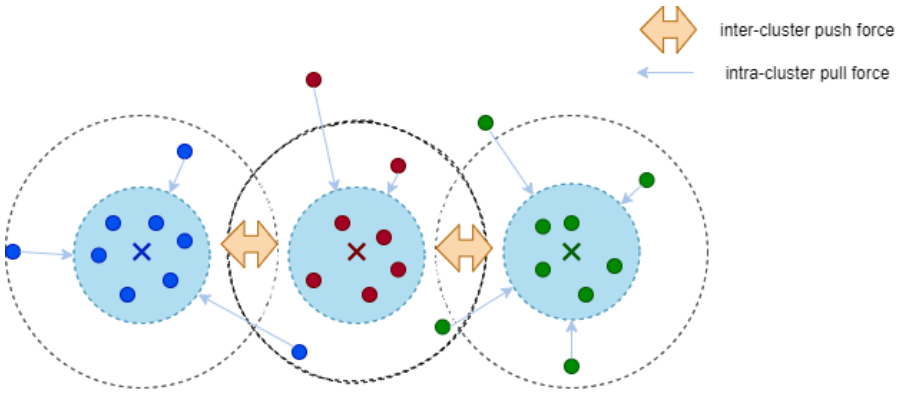


Figure 4.2: The intra-cluster pulling force pulls embeddings towards the cluster center, i.e. the mean embedding of that cluster. The inter-cluster repelling force pushes cluster centers away from each other.

After the network has converged, the embeddings of each lane pixels will be clustered together, resulting in at least δ_d apart between each cluster and no more than δ_v between embeddings and their cluster center.

4.1.3 Clustering

This part is finished by an iterative procedure. To select all embeddings belonging to the same lane, one can repeat taking a random lane embedding and thresholding around it with a bandwidth $b = 2\delta_v$ until all embeddings are assigned to a lane. In this case, thresholding means selecting all embeddings that lie within a hypersphere with radius b around the cluster centre. In order to avoid selecting an outlier to threshold around, mean shift is used first to shift closer to the cluster centre and then do the thresholding.

Chapter 5

LaneNet Quantization

This chapter introduces the quantization approach for the LaneNet. The key idea of this approach is from [19].

5.1 Background

5.1.1 Fixed-point Number Format

The main computation of the LaneNet are the convolution layers which can be simplified to a series of multiply-accumulate operations:

$$y = \sum_{i=1}^{K} w_i d_i \tag{5.1}$$

where w_i are the network parameters, d_i are data values, y is the output result of a single neuron before activation, and K is the kernel size.

LaneNet is generally trained in single-precision floating-point. To optimize this model, a fixed-point number representation which is a combination of Bit Width (BW), Integer Length (IL), Fractional Length (FL) and a sign bit:

$$BW_x = IL_x + FL_x + 1 \tag{5.2}$$

where x refers to a set of floating-point values that share the same fixed-point format.

A floating-point value x can be quantized to an integer by applying the follow formula:

$$Q(x) = round(x \cdot 2^{FL_x}) \tag{5.3}$$

It can be seen that the resolution of this format is 2^{-FL_x}. Then by scaling the integer value back, the value after quantized can be get:

$$\hat{x} = Q(x) \cdot 2^{-FL_x} \tag{5.4}$$

which should be within the following range:

$$-2^{IL_x} \leq \hat{x} \leq 2^{IL_x} - 2^{-FL_x} \tag{5.5}$$

To further illustrate this process, here we give an example (Figure 5.1). We set a fixed-point representation, in which $BW = 8$, $IL = 3$ and $FL = 4$. For this representation, it can represent the number in the range -8 (1000.0000) to 7.9375 (0111.1111). From this range, we take the number 3.4 as an example. The fractional part of the original binary representation of 3.4 (11.011001...) is apparently larger than 4. Therefore, after quantization, all the part that exceed the length limit

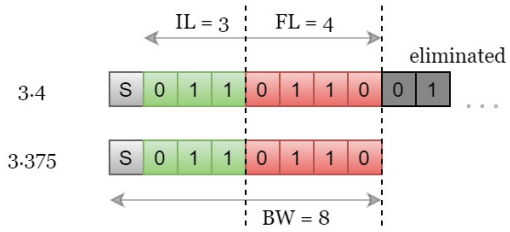


Figure 5.1: Example of fixed-point representation

will be eliminated. Then the number 3.4 will become 3.375 after quantization. By Equation 5.3 and 5.4, we can easily get this 3.375 ($round(3.4 \cdot 2^4) \cdot 2^{-4} = 3.375$).

To increase the resolution of this fixed-point format, the fractional length should be maximized. However, due to the bit width limited, this move will decrease the range. If data values fall outside the range, overflow will happen. Hence, it needs to be ensured that the integer length is chosen large enough to prevent overflow:

$$IL_x = \lfloor log_2(R_x) \rfloor + 1 \tag{5.6}$$

where range R_x is the absolute maximum value within the group value of $\mathbf{x}$:

$$R_{\mathbf{x}} = max_{x \in \mathbf{x}} |x| \tag{5.7}$$

For example, consider a fixed-point group with $BW = 16$ and $max(|x|) = 1.154438$, then the minimum integer length should be $IL_x = \lfloor log_2(1.154438) \rfloor + 1 = 1$ and the maximum fractional length is $FL_x = BW_x - IL_x - 1 = 14$.

5.1.2 Constraint

As is mentioned before, the main computation of convolutional layer can be simplified to a series of dot products, shown in equation 5.1. It may occur overflow between the multiplications and additions between the input data and the weights. Like Figure 5.2 shows, the result after multiplying an input data 3.375 ($BW_d = 8$) and a weight 0.625 ($BW_w = 5$) requires more than BW_d and BW_w to store it. Therefore, a constraint here need to be discussed.

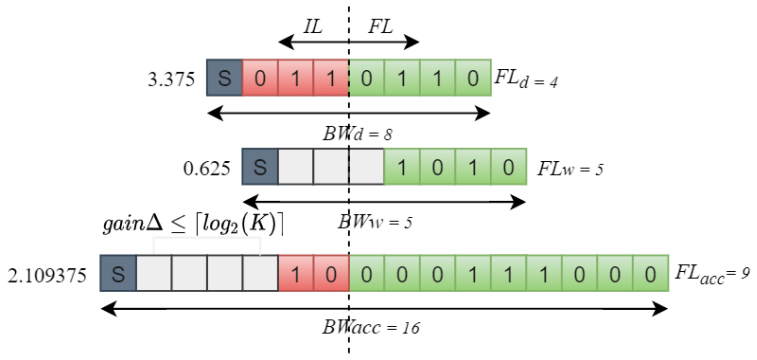


Figure 5.2: Example of fixed-point multiply-accumulation

To prevent overflow in this computation, the required number of bits can be computed:

$$BW_{w \times d} = (IL_w + IL_d + 1) + (FL_w + FL_d) + 1 = BW_W + BW_d \tag{5.8}$$

Excluding the case that the most negative values of both operands are multiplied i.e. $-2^{IL_d} \cdot -2^{IL_d}$ for gaining one bit of additional precision, the resulting bit width becomes:

$$BW_{w \times d} = BW_w + BW_d - 1 \tag{5.9}$$

This result is limited by the size of accumulation register which will be large enough to fit the multiplication result without overflowing:

$$BW_w + BW_d - 1 \leq BW_{acc} \tag{5.10}$$

Furthermore, to prevent overflow during kernel computation, the accumulation bit width must be large enough to fit the final accumulation result. The worst-case required number of bits for K accumulations of the product is:

$$(BW_d + BW_w - 1) + \lceil log_2(K) \rceil \leq BW_{acc} \tag{5.11}$$

For example, if we have a kernel with 4 integer weights ($BW_d = BW_w = 8$): $w_1 = 99, w_2 = 76, w_3 = 38, w_4 = 77$. We cannot safely make assumptions about the magnitude and sign of input data, other than the worst-case -2^{BW_d-1}. Therefore, the maximum output value would be $(99 + 76 + 38 + 77) \times 2^7 = 37120$ which requires 17 bit including the sign bit. This fits in the constraint $(8 + 8 - 1) + log_2 4 = 17$.

5.2 Quantization Procedure

The basic procedure of quantization is depicted in Figure 5.3 below.

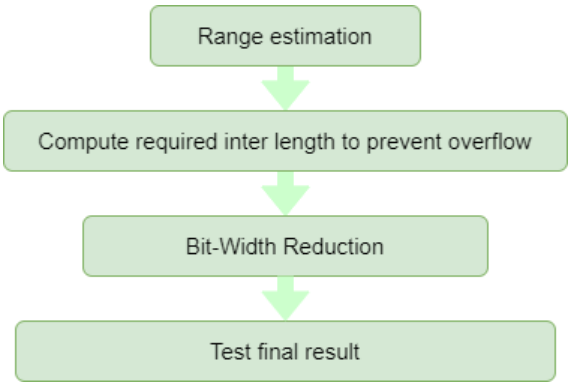


Figure 5.3: Quantization procedure

- Range Analysis
  The first step is to analyse the parameter and input data range $R_{\mathbf{x}}$ to determine integer lengths IL_w and IL_d. The precision FL_w and FL_d will be determined by the available bit widths BW_w and BW_d (see Equation 5.2). Due to the different ranges of these two data groups, using single format for both of them is not preferred. The input data range can be estimated by forwarding a large batch of images from the test dataset through the network and using Equation 5.6, the required integer length for input can be determined. And the integer length for weights can be get at compiling time.

- Quantization
  The next step is to perform a binary search to find the optimal number of bits for convolutional weights and convolutional inputs. In this step, a certain network part is quantized, while the rest remains in floating-point. Since the inputs and weights of convolutional layers need independent bit widths, iteratively quantizing one network part can find the optimal bit width for each part.

Algorithm 1 below illustrates this process in detail. It will first run forward a number of image batches through the network, resulting in the group values of input x_d^l, weight x_w^l and output

x^l_{out} by layers, as well as the baseline accuracy acc_{ref}. Then, the maximum value of x^l_d, x^l_w and x^l_{out} can be got, and further, by the range analysis, the integer length of input il^l_d, weight il^l_w and output il^l_{out} can be calculated. After that, we set the initial quantized bit width of input and weight, bw_d and bw_w. Once we have the integer length and the total bit width, the fractional length (fl^l_d, fl^l_w) can be calculated by $bw - il^l - 1$. After getting the kernel size K, the accumulator bit width can be set by $bw_d + bw_w - 1 + \lceil log_2 K \rceil$. After we had all parameters required, the original convolutional layers can be changed into the quantized convolution layers. By running forwards numbers of image batches, the accuracy after quantization acc can be got. As long as the acc is not lower than a certain level, the bit width of input and data will decrease by two and the quantization process will be done again. When the resulting acc drop by a certain level that we can not accept, the final bit width of input and weight ($best_bwd$, $best_bw_w$) can be found. Once a good trade-off between small number representation and classification accuracy is found, the resulting fixed-point network is obtained.

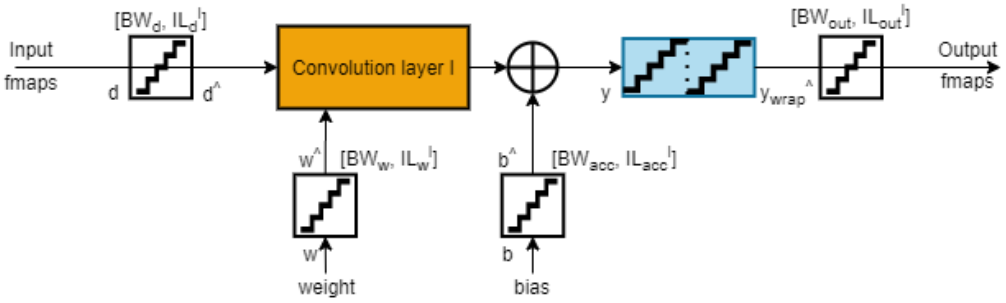


Figure 5.4: Simulate fixed-point convolution layer during forward pass

Figure 5.4 shows a simulation of a fixed-point data path with limited precision. Several quantization operators are inserted before and after every convolution layer, making the reduction of precision for a subset of layers very easy. These uniform quantizers map the high-precision input data, weights and accumulator values to a discrete space. During the forward pass, these quantizers will reduce the precision of fixed-point groups according to Equation 5.4. Range of input data and weight is clipped within representable range as in Equation 5.5 using the integer lengths found during range analysis. Also to make sure the range of the layer output is sitting in Equation 5.5, as well as to prevent overflow, simply clipping the range may lead to serious error. Therefore, a wrap-around operator, according to the BW_{acc} calculated by Equation 5.11 above, is simulated after layer output $\hat{y}$ to ensure correct operation:

$$\hat{y}_{wrap} = y_{min} + (y_{min} + \hat{y}) \% y_{range} \tag{5.12}$$

in which $y_{min} = -2^{BW_{acc}-(FL_d+FL_w)-1}$, $y_{range} = 2^{BW_{acc}-(FL_d+FL_w)}$, and the modulo operator computes the remainder using floored division. Here we also take Figure 5.2 as an example. We have the input data with $BW_d = 8$ ($FL_d = 4$) and the weight with $BW_w = 6$ ($FL_w = 5$). We assume the BW_{acc} here is 17. Then we can get the $y_{min} = -2^{17-(4+5)-1} = -2^7 = -128$, and the $y_{range} = 2^{17-(4+5)} = 2^8 = 256$. If we now have an output y sit outside the range, such as 2^9, then it will be wrap around using the Equation 5.12 to 2^{-7}.

Algorithm 1 Quantization procedure to find the optimal number of bits for convolutional inputs and weights. The LaneNet that is being quantized is denoted by *net*, which consists of a set of layer l. For the quantization here, the convolution layer *conv_layer* is the only considered layer.

Input: the network *net*, error margin $error\_margin = 5$, maximum bitwidth $max\_bw = 16$
Output: the best bitwidth for the input data and weight $best\_bw_d$, $best\_bw_w$

Running forwards a number of image batches through *net*, get the baseline accuracy acc_ref and the group value of input x_d^l, the group value of weight x_w^l, the group value of x_{out}^l
for all $l \in conv\_layer$ **do**
  get the absolute max value of the input data $max_d^l \leftarrow max_{x_d \in x_d^l} |x_d|$
  set the integer length of the input data $il_d^l \leftarrow \lfloor log_2(max_d^l) \rfloor + 1$
  get the absolute max value of the weight $max_w^l \leftarrow max_{x_w \in x_w^l} |x_w|$
  set the integer length of the weight $il_w^l \leftarrow \lfloor log_2(max_w^l) \rfloor + 1$
  get the absolute max value of the output data $max_{out}^l \leftarrow max_{x_{out} \in x_{out}^l} |x_{out}|$
  set the integer length of the output data $il_{out}^l \leftarrow \lfloor log_2(max_{out}^l) \rfloor + 1$
end for
set the initial quantized input data bitwidth $bw_d \leftarrow 16$
while $bw_d > 0$ **do**
  for all $l \in conv\_layer$ **do**
    get the kernel size K
    set the accumulator bitwidth $bw\_acc \leftarrow (2 \cdot max\_bw - 1) + \lceil log_2(K) \rceil$
    set the floating length of input data $fl_d^l \leftarrow bw_d - il_d^l - 1$
    get the accuracy *acc* from forwarding image batches through validation dataset
  end for
  if $acc + error\_margin/100 < acc\_ref$ **then**
    get the best bitwidth for the input data $best\_bw_d \leftarrow bw_d + 2$
    break
    $bw_d \leftarrow bw_d - 2$
  end if
end while
set the initial quantized weight bitwidth $bw_w \leftarrow 16$
while $bw_w > 0$ **do**
  for all $l \in conv\_layer$ **do**
    get the kernel size K
    set the accumulator bitwidth $bw\_acc \leftarrow (2 \cdot max\_bw - 1) + \lceil log_2(K) \rceil$
    set the floating length of input data $fl_w^l \leftarrow bw_w - il_w^l - 1$
    get the accuracy *acc* from forwarding image batches through validation dataset
  end for
  if $acc + error\_margin/100 < acc\_ref$ **then**
    get the best bitwidth for the weight $best\_bw_w \leftarrow bw_w + 2$
    break
  end if
  $bw_w \leftarrow bw_w - 2$
end while
return $best\_bw_d$, $best\_bw_w$

Chapter 6

Experimental Setup

6.1 Experimental Setup

Figure 6.1 depicts the inside sturcture of the LaneNet. LaneNet's architecture is based on the network VGG16 [52], which is consequently modified into a two-branched network. The last layer of the embedding branch outputs an N-channel image, where the N is the embedding dimension. The last layer of the segmentation branch outputs a one channel image (binary segmentation). LaneNet is trained with an embedding dimension of 7, with $\delta_v = 0.5$ and $\delta_d = 3$. The images are rescaled to 512×288 and the network is trained with a learning rate $1e-2$ until convergence. The training process is based on Pytorch 1.5.0 with CUDA 10.0.

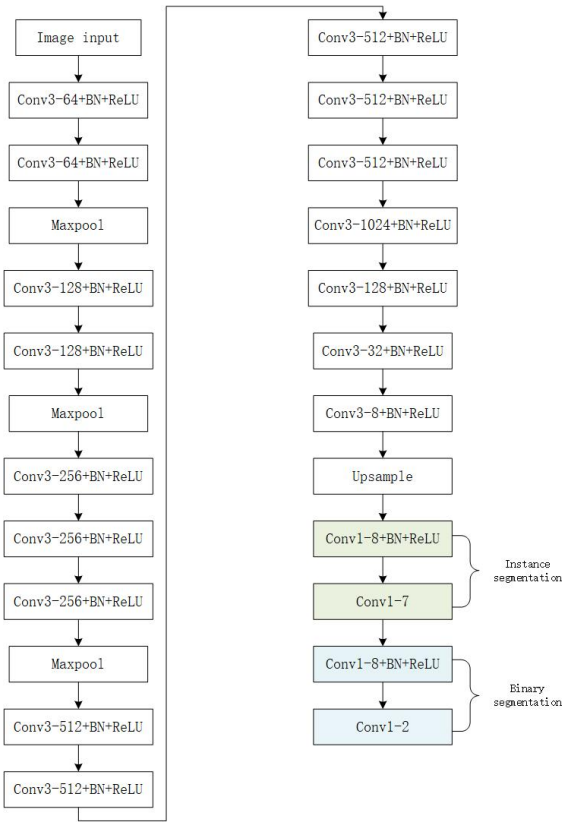


Figure 6.1: Structure of LaneNet

6.2 Dataset

6.2.1 TuSimple

The network is trained using the TuSimple lane dataset [53] which is the only large scale dataset for testing deep learning approaches on the lane detection task. The dataset consists of 3626 training and 2782 testing video clips, which are recorded on 2-lane/3-lane/4-lane/or more highway roads at different day time, under different traffic condition and both good and medium weather conditions. Each video clip contains total 20 image frames, which are not annotated. Figure 6.2 below shows some examples of the TuSimple dataset. The annotations are in a JSON file, indicating the x-position of the lanes at a number of discretized y-position. When changing lane, there can be an extra 5th lane added to avoid confusion. An example of the format of each line in the JSON file is shown below. All the lanes on each image are around the center of sight, encouraging the autonomous vehicle to focus on the current lane and left/right lanes.

```
{
    'raw_file': str. 20th frame file path in a clip.
    'lanes': list. A list of lanes. For each list of one lane, the elements are
        width values on image.
    'h_samples': list. A list of height values corresponding to the 'lanes',
        which means len(h_samples) == len(lanes[i])
}
```

(a) (b) (c)

(d) (e) (f)

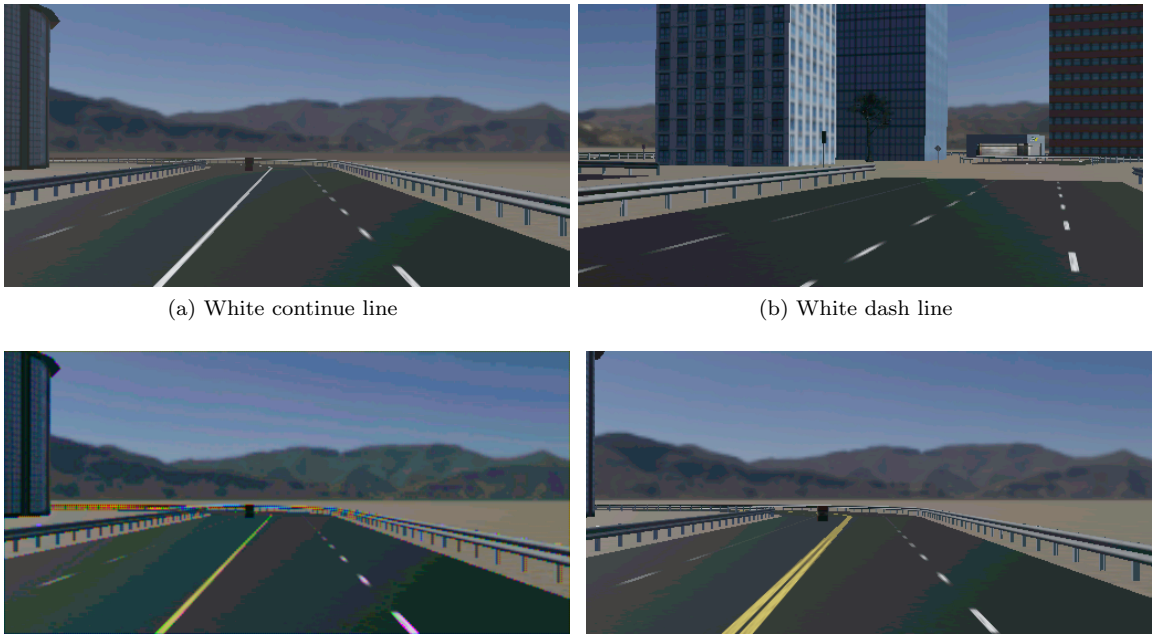
(g) (h) (i)

Figure 6.2: Examples of TuSimple

6.2.2 Custom dataset

The network is also trained using the custom dataset which is created by [54] using Webots according to the format of the TuSimple dataset. The images contain various environment scenarios. For

instance, different types of lanes with various colors and forms, different layout of road, different scenes or weathers. Figure 6.3, 6.4, 6.5 show some of the examples.



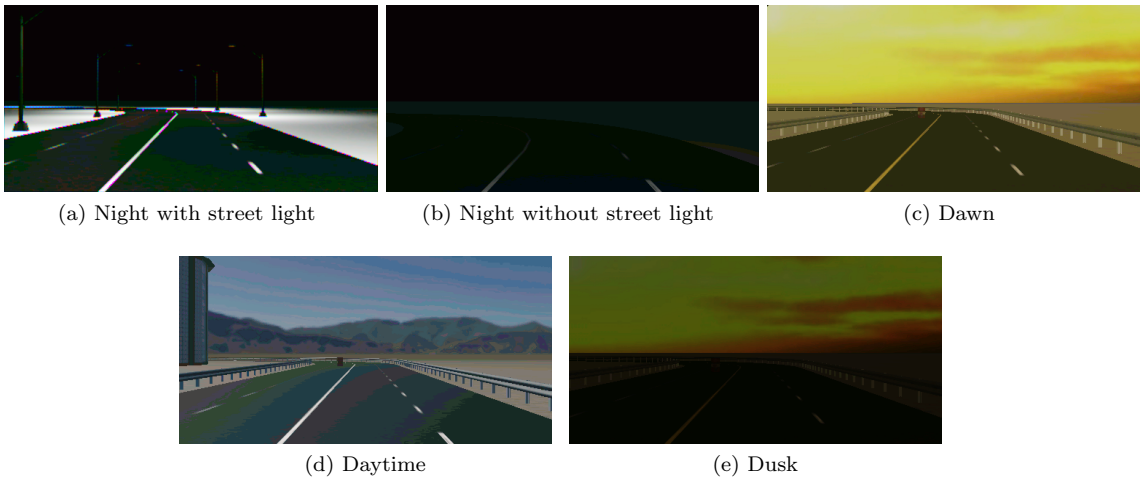
(a) White continue line

(b) White dash line

(c) Yellow continue line

(d) Yellow double line

Figure 6.3: Examples of different lane types in custom dataset



(a) Night with street light

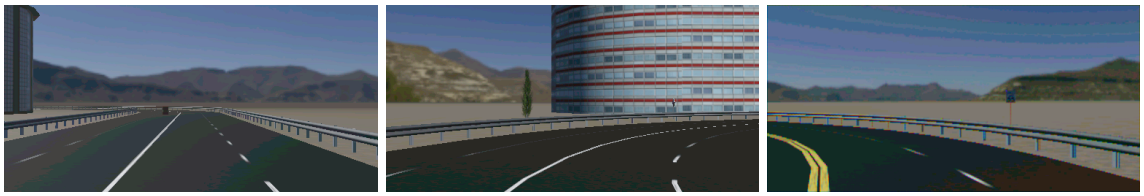
(b) Night without street light

(c) Dawn

(d) Daytime

(e) Dusk

Figure 6.4: Examples of different weathers in custom dataset



(a) Straight (b) Right turn (c) Left turn

Figure 6.5: Examples of different road layout in custom dataset

6.2.3 Evaluation

To measuring the performance of the model, the accuracy is calculated as the average correct number of points per image:

$$acc = \sum_{im} \frac{C_{im}}{S_{im}} \tag{6.1}$$

where C_{im} is the number of correct points and S_{im} is the number of ground-truth points. A point is correct when the difference between a ground-truth and predicted point is less than a certain threshold.

Chapter 7

Results and Discussions

7.1 Quantization of the LaneNet with TuSimple dataset

In this section, the results of approximating 32-bit floating point LaneNet by condensed fixed point model are presented. All classification accuracy were obtained running the respective network on the whole TuSimple dataset. The baseline 32-bit floating point LaneNet is trained for 50 epoch which has an accuracy of 89.412%. Figure 7.1 shows an example of the output of the trained LaneNet testing with the TuSimple test dataset. Table 7.1 below shows the results of iteratively quantizing only weight, only input data and both from 16 bit to 6 bit. Basically, the accuracy is decreasing when reducing the bit width. Figure 7.4 below shows the comparison of the three results It can be seen from the Pareto diagram that the bit width of 8 for both weight and input data gives a good trade-off with an error of 1.97% between representation and accuracy. Therefore, for the resulting fixed-point model, 8 bits is chosen to be the optimal bit width for both weights and inputs.

Table 7.1: The result of iteratively quantizing weight and input data with TuSimple dataset

LaneNet with TuSimple dataset, 32-bit floating point accuracy: 89.412%					
Only quantized weight	Accuracy	Only quantized input	Accuracy	Quantized both weight and input	Accuracy
16	89.415%	16	89.406%	16	89.405%
14	89.413%	14	89.408%	14	89.409%
12	89.392%	12	89.373%	12	89.392%
10	89.336%	10	89.269%	10	89.100%
8	87.581%	8	85.515%	8	87.649%
6	56.897%	6	7.010%	6	1.877%

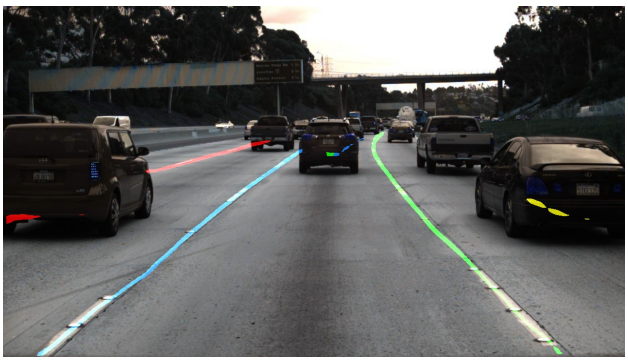


Figure 7.1: The output of the trained LaneNet with test TuSimple dataset

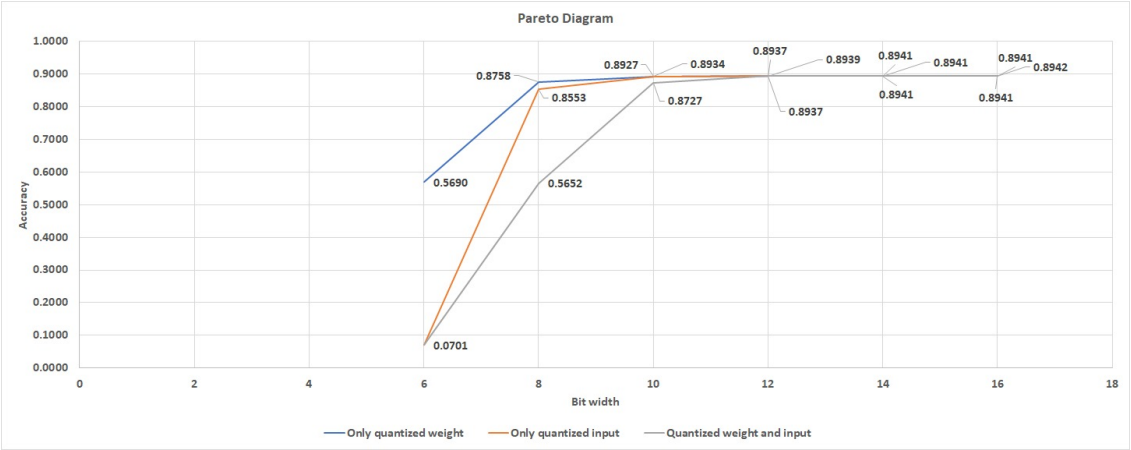


Figure 7.2: The Pareto diagram of only quantized weight and only quantized input with TuSimple dataset

7.2 Quantization of the LaneNet with custom dataset

In this section, the result of quantizing the LaneNet that trained by the custom dataset is shown in Table 7.2. The baseline 32-bit floating point LaneNet, is trained for 30 epoch and has an accuracy of 99.417%. Figure 7.2 shows an example of the output of the trained LaneNet testing with the custom test dataset. The overall accuracy rate decreases as the number of bits decreases. It can be seen that, from 16bit to 12bit, there is almost no change in the result, while starting from 6bit, the accuracy rate drops sharply. Therefore, for the resulting fixed-point model, 8 bits is chosen to be the optimal bit width for both weights and inputs. It has an acceptable accuracy of 98.774% with an error of 0.65% comparing to the original floating point model.

Table 7.2: The result of iteratively quantizing weight and input data with custom dataset

LaneNet with custom dataset, 32-bit floating point accuracy: 99.417%					
Only quantized weight	Accuracy	Only quantized input	Accuracy	Quantized both weight and input	Accuracy
16	99.417%	16	99.416%	16	99.417%
14	99.417%	14	99.417%	14	99.417%
12	99.416%	12	99.417%	12	99.414%
10	99.422%	10	99.412%	10	99.421%
8	99.354%	8	98.592%	8	98.774%
6	57.109%	6	46.655%	6	31.966%

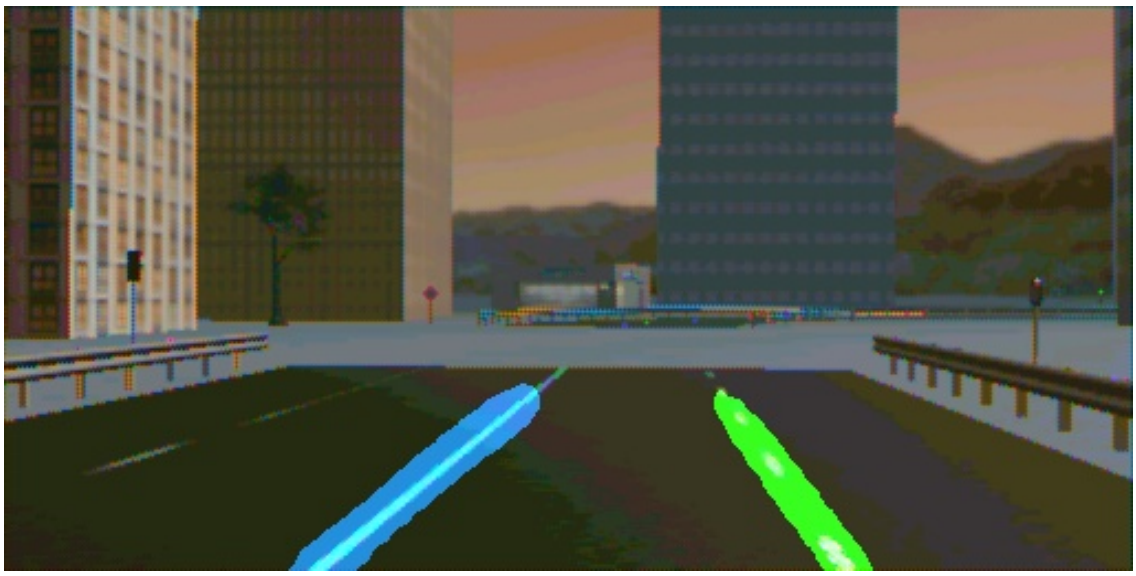


Figure 7.3: The output of the trained LaneNet with test custom dataset

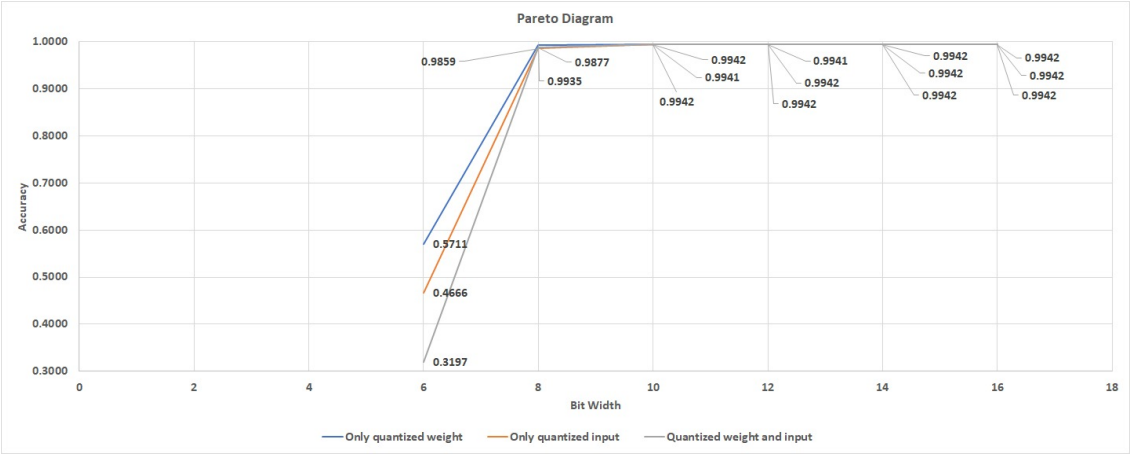


Figure 7.4: The Pareto diagram of only quantized weight and only quantized input with custom dataset

7.3 The comparison of the results of two datasets

From both Figure 7.4 and Figure 7.2, it can be easily seen that the accuracy keeps a downward tendency as the precision declines. From 16 bit to 10 bit, the accuracy almost remains the same without much differences. However, in both figures, at 8 bit, there is a relatively significant drop in accuracy and at 6 bit, a drastic decrease of the accuracy makes the model unable to complete the classification tasks. Hence, in general, the limiting bit width of this quantization method for LaneNet is 8 bit. Further, comparing the three results of only quantized weight, only quantized input and quantized both input and weight, it can be seen that the accuracy of only quantized weight is always the highest while the accuracy of quantized both input and weight is always the lowest. This is probably because the number of parameters is much higher than the number of input which enables it to have better robustness, leading to less reduction in accuracy. Whereas, when the precision of both input and weight go down, it's not hard to imagine that the accuracy will drop the most. In conclusion, 8 bit is the limitation of this quantization method on LaneNet, which has the best trade-off between accuracy and the degree of model compression.

Chapter 8

Conclusions and Future Works

8.1 Conclusion

Camera-based lane detection algorithms have already been developed by using CNNs, which replaces the hand-crafted features leading to better robustness to the variation of the environments. It has been found that the CNNs for large scale image recognition always face the issue of over-parameterized. As to compress the network with a tolerable accuracy penalty, quantization, converting the floating-point parameters to a fixed-point representation is a potential solution.

In this paper, we trained, validated and tested the lane detection model LaneNet with TuSimple and custom datasets, getting the accuracy of 89.412% and 99.417% respectively. Then we quantized the floating-point lane detection model LaneNet to fixed-point. The result of the network trained by both datasets demonstrates that with the decrease of the precision of the network, the accuracy always keeps a downward tendency. For both cases, 8 bit seems the best choice which has the best trade-off between accuracy and the degree of model compression. In the TuSimple case, the network at 8 bit achieved an accuracy of 87.649%, leading to a 1.97% error comparing to the original baseline accuracy. And in the custom case, the network at 8 bit even reached an accuracy of 99.354% with an error of 0.65% comparing to the base line accuracy.

8.2 Future work

There may be several works that can be done in the future.

- Except for the quantization method talked about in this paper, there are also other model compression methods, such as parameter pruning, low-rank decomposition, knowledge distillation, etc, which could be discussed. Maybe by comparing these methods or even combining these methods, a more efficient model compression method can be found.
- Expect the accuracy, there are still other aspects influenced by the quantization, for instance memory optimization and speed optimization, which could be discussed.

Bibliography

[1] Lane Keeping assistant. https://www.autoevolution.com/news/lane-keeping-assist-systems-explained-25459.html. v, 1

[2] Adaptive cruise control. https://www.audi-mediacenter.com/en/photos/detail/adaptive-cruise-control-42099. v, 1

[3] Automatic parking. https://www.milescontinental.co.nz/news/features/volkswagen-park-assist-automatic-parking/. v, 1

[4] Iuliana Tabian, Hailing Fu, and Zahra Sharif Khodaei. A convolutional neural network for impact detection and characterization of complex composite structures. *Sensors*, 19(22), 2019. v, 2

[5] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S. Emer. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 105(12):2295–2329, 2017. v, 9

[6] Santokh Singh. Critical reasons for crashes investigated in the national motor vehicle crash causation survey. Technical report, 2015. 1

[7] Sayandip De, Sajid Mohamed, Dip Goswami, and Henk Corporaal. Approximation-aware design of an image-based control system. *IEEE Access*, 2020. 1

[8] Sayandip De, Sajid Mohamed, Konstantinos Bimpisidis, Dip Goswami, Twan Basten, and Henk Corporaal. Approximation trade offs in an image-based control system. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1680–1685. IEEE, 2020. 1

[9] Sajid Mohamed, Sayandip De, Konstantinos Bimpisidis, Vishak Nathan, Dip Goswami, Henk Corporaal, and Twan Basten. IMACS: a framework for performance evaluation of image approximation in a closed-loop system. In *MECO*, 2019. 1

[10] Sajid Mohamed, Dip Goswami, Vishak Nathan, Raghu Rajappa, and Twan Basten. A scenario-and platform-aware design flow for image-based control systems. *Microprocessors and Microsystems*, 2020. 1

[11] Sajid Mohamed, Nilay Saraf, Daniele Bernardini, Dip Goswami, Twan Basten, and Alberto Bemporad. Adaptive predictive control for pipelined multiprocessor image-based control systems considering workload variations. In *59th IEEE Conference on Decision and Control (CDC)*, 2020. 1

[12] Sajid Mohamed, Asad Ullah Awan, Dip Goswami, and Twan Basten. Designing image-based control systems considering workload variations. In *IEEE 58th Conference on Decision and Control (CDC)*, pages 3997–4004. IEEE, 2019. 1

[13] Yang Xing, Chen Lv, Long Chen, Huaji Wang, Hong Wang, Dongpu Cao, Efstathios Velenis, and Fei-Yue Wang. Advances in vision-based lane detection: Algorithms, integration, assessment, and perspectives on acp-based parallel vision. *IEEE/CAA Journal of Automatica Sinica*, 5(3):645–661, 2018. 1

[14] Bor-Yiing Su. *Parallel Application Library for Object Recognition*. PhD thesis, EECS Department, University of California, Berkeley, Sep 2012. 2

[15] Misha Denil, Babak Shakibi, Laurent Dinh, Marc'Aurelio Ranzato, and Nando de Freitas. Predicting parameters in deep learning, 2014. 2

[16] Yeongmin Ko, Younkwan Lee, Shoaib Azam, Farzeen Munir, Moongu Jeon, and Witold Pedrycz. Key points estimation and point instance segmentation approach for lane detection, 2020. 2

[17] Yuenan Hou, Zheng Ma, Chunxiao Liu, and Chen Change Loy. Learning lightweight lane detection cnns by self attention distillation, 2019. 2

[18] Davy Neven, Bert De Brabandere, Stamatios Georgoulis, Marc Proesmans, and Luc Van Gool. Towards end-to-end lane detection: an instance segmentation approach, 2018. 2, 8, 11

[19] Barry de Bruin, Zoran Zivkovic, and Henk Corporaal. Quantization of deep neural networks for accumulator-constrained processors. *Microprocessors and Microsystems*, 72, February 2020. 4, 13

[20] T. B. Preußer, G. Gambardella, N. Fraser, and M. Blott. Inference of quantized neural networks on heterogeneous all-programmable devices. In *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 833–838, 2018. 4

[21] LDKS. `https://web.archive.org/web/20170701082928/http://usedsemitrailers.com/lane-departure-warning-system/`. 6

[22] Honda LKAS. `https://web.archive.org/web/20141230004825/http://world.honda.com/news/2003/4030618_2.html`. 6

[23] Honda full briefing. `http://www.hondauk-media.co.uk/uploads/presspacks/5a02fe90161c72db29ce5d84babed2045eb84c6b/Full_Briefing.pdf`. 6

[24] Lexus lkas. `https://web.archive.org/web/20070222021654/http://www.lexus-europe.com/lexus_cars/ls/ls460/showcase/index.asp`. 6

[25] Audi Q7. `https://www.audiworld.com/articles/the-audi-q7-4-2-tdi/`. 6

[26] Tesla LKAS. `https://www.greencarreports.com/news/1094773_tesla-model-s-adds-speed-assist-lane-departure-warning`. 6

[27] Pilot Assistant. `http://volvo.custhelp.com/app/answers/detail/a_id/9731/~/pilot-assist-ii`. 6

[28] Nissan LKS. `https://web.archive.org/web/20050110073214/http://ivsource.net/archivep/2001/feb/010212_nissandemo.html`. 6

[29] Yue Wang, Eam Khwang Teoh, and Dinggang Shen. Lane detection and tracking using b-snake. *Image and Vision computing*, 22(4):269–280, 2004. 6

[30] M. Bertozzi and A. Broggi. Gold: a parallel real-time stereo vision system for generic obstacle and lane detection. *IEEE Transactions on Image Processing*, 7(1):62–81, 1998. 6

[31] Trung-Thien Tran, Chan-Su Bae, Young-Nam Kim, Hyo-Moon Cho, and Sang-Bock Cho. An adaptive method for lane marking detection based on hsi color model. In *International Conference on Intelligent Computing*, pages 304–311. Springer, 2010. 7

[32] K. Kluge and S. Lakshmanan. A deformable-template approach to lane detection. In *Proceedings of the Intelligent Vehicles '95. Symposium*, pages 54–59, 1995. 7

[33] Felix Măriut, Cristian Foşalău, and Daniel Petrisor. Lane mark detection using hough transform. In *2012 International Conference and Exposition on Electrical and Power Engineering*, pages 871–875, 2012. 7

[34] Amol Borkar, Monson Hayes, and Mark T Smith. A novel lane detection system with efficient ground truth generation. *IEEE Transactions on Intelligent Transportation Systems*, 13(1):365–374, 2011. 7

[35] R. Gopalan, T. Hong, M. Shneier, and R. Chellappa. A learning approach towards detection and tracking of lane markings. *IEEE Transactions on Intelligent Transportation Systems*, 13(3):1088–1098, 2012. 7

[36] Jihun Kim and Minho Lee. Robust lane detection based on convolutional neural network and random sample consensus. In Chu Kiong Loo, Keem Siah Yap, Kok Wai Wong, Andrew Teoh, and Kaizhu Huang, editors, *Neural Information Processing*, pages 454–461, Cham, 2014. Springer International Publishing. 7

[37] Brody Huval, Tao Wang, Sameep Tandon, Jeff Kiske, Will Song, Joel Pazhayampallil, Mykhaylo Andriluka, Pranav Rajpurkar, Toki Migimatsu, Royce Cheng-Yue, Fernando Mujica, Adam Coates, and Andrew Y. Ng. An empirical evaluation of deep learning on highway driving, 2015. 7

[38] Bei He, Rui Ai, Yang Yan, and Xianpeng Lang. Accurate and robust lane detection based on dual-view convolutional neutral network. In *2016 IEEE Intelligent Vehicles Symposium (IV)*, pages 1041–1046, 2016. 7

[39] J. Li, X. Mei, D. Prokhorov, and D. Tao. Deep neural network for structural prediction and lane detection in traffic scene. *IEEE Transactions on Neural Networks and Learning Systems*, 28(3):690–703, 2017. 7

[40] Lucas Tabelini, Rodrigo Berriel, Thiago M. Paixão, Claudine Badue, Alberto F. De Souza, and Thiago Oliveira-Santos. Polylanenet: Lane estimation via deep polynomial regression, 2020. 8

[41] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. Deep learning with limited numerical precision. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, page 1737–1746. JMLR.org, 2015. 8

[42] Yufei Ma, Naveen Suda, Yu Cao, Jae-sun Seo, and Sarma Vrudhula. Scalable and modularized rtl compilation of convolutional neural networks onto fpga. In *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–8, 2016. 8

[43] Philipp Gysel, Mohammad Motamedi, and Soheil Ghiasi. Hardware-oriented approximation of convolutional neural networks, 2016. 8

[44] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations, 2016. 8

[45] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1, 2016. 8

[46] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks, 2016. 8

[47] Zhaowei Cai, Xiaodong He, Jian Sun, and Nuno Vasconcelos. Deep learning with low precision by half-wave gaussian quantization, 2017. 9

[48] Fengfu Li, Bo Zhang, and Bin Liu. Ternary weight networks, 2016. 9

[49] Chenzhuo Zhu, Song Han, Huizi Mao, and William J. Dally. Trained ternary quantization, 2017. 9

[50] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Training deep neural networks with low precision multiplications, 2015. 10

[51] Bert De Brabandere, Davy Neven, and Luc Van Gool. Semantic instance segmentation with a discriminative loss function, 2017. 12

[52] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015. 18

[53] Tusimple. `https://github.com/TuSimple/tusimple-benchmark/tree/master/doc/lane_detection`. 19

[54] Sayandip De, Yingkai Huang, Sajid Mohamed, Dip Goswami, and Henk Corporaal. Hardware- and situation-aware sensing for robust closed-loop control systems. 2021. Design, Automation and Test in Europe Conference 2021 ; Conference date: 01-02-2021 Through 05-02-2021. 19