Low Cost and Power CNN/Deep Learning Solution for Automated Driving

Mihir Mody, Desappan Kumar, Pramod Swami, Manu Mathew and Soyeb Nagori
Automotive Processor, Texas Instruments Incorporated, Bangalore, India
E-mail: {mihir, kumar.desappan, pramods, mathew.manu, soyeb}@ti.com

Abstract

Automated driving functions, like highway driving and parking assist, are increasingly getting deployed in high-end cars with the ultimate goal of realizing self-driving car using Deep learning techniques like convolution neural network (CNN). For mass-market deployment, the embedded solution is required to address the right cost and performance envelope along with security and safety. In the case of automated driving, one of the key functionality is "finding drivable free space", which is addressed using deep learning techniques like CNN. These CNN networks pose huge computing requirements in terms of hundreds of GOPS/TOPS (Giga or Tera operations per second), which seems beyond the capability of today's embedded SoC. This paper covers various techniques consisting of fixed-point conversion, sparse multiplication, fusing of layers and network pruning, for tailoring on the embedded solution. These techniques are implemented on the device by means of optimized Deep learning library for inference. The paper concludes by demonstrating the results of a CNN network running in real time on TI's TDA2X embedded platform producing a high-quality drivable space output for automated driving.

Keywords

Automated Driving, Deep Learning, CNN, semantic segmentation, TDA processor

1. Introduction

Automated driving functions, like highway driving and parking assist, are increasing getting deployed in high-end cars with the ultimate goal of realizing self-driving car. To help the evolution of these functions various level (1-5) are defined by automotive standardization organizations [1].

Figure 2 shows the logical block diagram of automated driving functions. The key blocks for automated driving are Perception, Localization, Fusion, Driving Policy, Motion Planning, and Control. The multi-modality perception (Camera, radar, and Lidar) is used to gather environment information around the vehicle. Fusion module is used give most reliable environment (e.g. Bayesian filtering) among all modalities. The localization module is used to find the exact position of the vehicle in real-world co-ordinates using HD Maps and perception data. The resulting environment model is used by Driving Policy module to take decision e.g. stay in lane, lane change, yield, merge etc. The decision of Driving policy module is translated into actual car movement with Motion planning module accounting kinematics and passenger's comfort. Lastly, control (e.g. PID Control) is used to track actual vehicle trajectory with reference by controlling actuators. The Deep learning techniques consisting of Convolution Neural Network (CNN), Recurrent Neural networks (RNN), Deep Reinforcement learning (DRL) are used extensively across all modules to achieve the goal of the self-driving car. It is the de-facto method for camera perception module, while is used along with traditional computer vision and machine learning algorithms to enhance performance.

Typical Convolution Neural Network (CNN) structure is illustrated as shown in Figure 1. Input feature vectors are convolved with a set of pre-trained receptive field weights followed by a non-linear activation function. Max-pooling enables translation invariance and reduces the output feature vector size. The learned output feature vector is fed to the fully connected neural network for classification, with Softmax layer normalizes the results. There are multiple network topologies e.g. AlexNet[2], VGGNet[3], GoogleNet etc[4]. These networks have multiple convolution layers and fully connected layers, which results in huge compute complexity going in hundreds of Giga or Tera Multiply and Add operations (GOPS or TOPS) with 2D convolution function taking more than 95% of the overall computation. These network architectures have evolved without regard to model complexity and computational efficiency. On the other hand, successful deployment of CNNs on embedded platforms requires small model sizes to accommodate limited on-device memory and real-time execution with minimal accuracy loss.

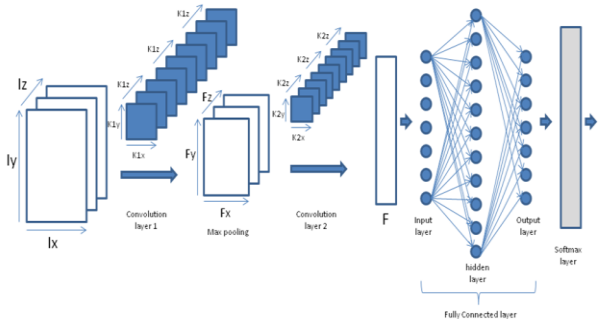

Figure 1 A typical Convolution Neural Network with two convolution layers and fully connected layer followed by softmax.

The section 2 gives the overview of various techniques to optimize CNN on the embedded platform. The section 3 deep dives into exact mapping and usage for free space detection using semantic segmentation algorithm on Texas Instrument's TDA2 [14] automotive processor. The Section 4 gives details of experiments and results with section 5 concluding the paper.

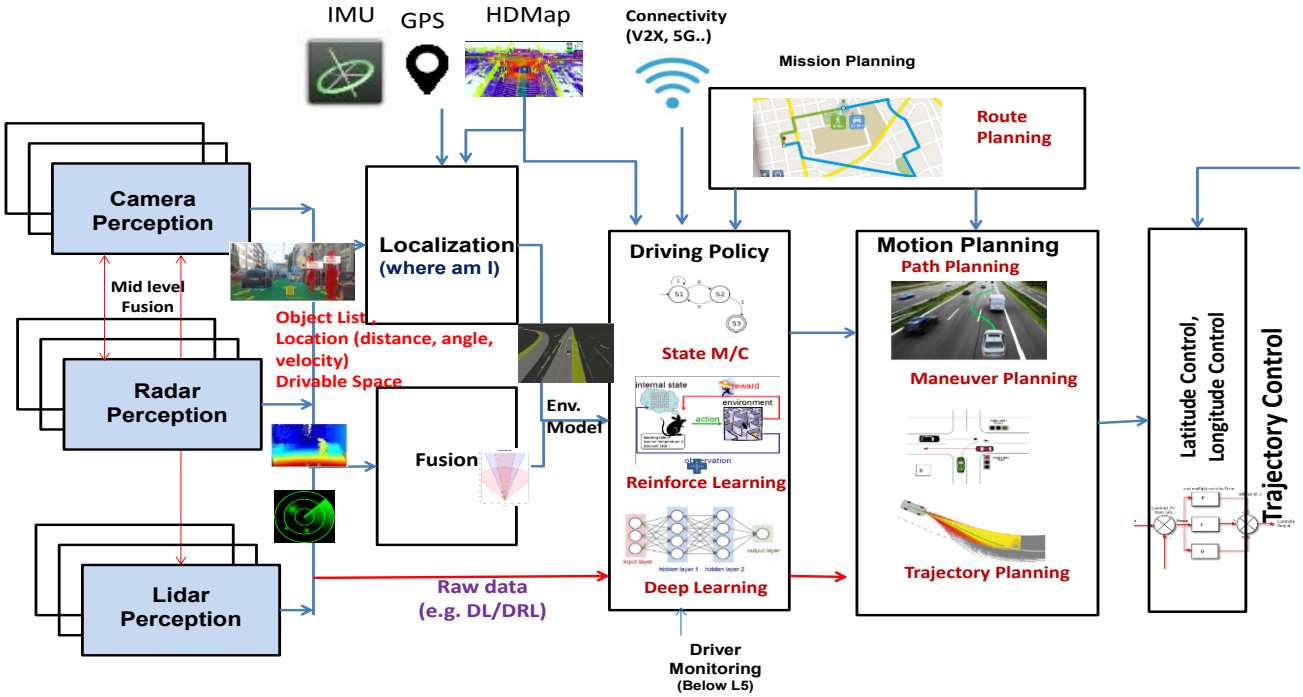

Figure 2 Automated Driving Block Diagram

2. Techniques for Embedded Platform

This paper covers various techniques consisting of fixed-point conversion, sparse multiplication, fusing of layers and network pruning, to enable low cost embedded solution.

2.1 Fixed Point CNN Inference

Successful deployment of CNNs on embedded platforms requires small model sizes to accommodate limited on-device memory and real-time execution. This has led to a growing field of research that focuses on quantized CNN inference for embedded devices having limited resources with minimal accuracy losses. The CNN quantization methods in general fall into two categories namely Training/Fine-tuning CNN models accounting quantization [5, 6, 8, 9] process and Quantized CNN inference with off-line floating point to fixed point conversion [7, 10, 11].

2.1.1 Training/Fine-tuning CNN model

This method accounts quantization during the training process on PC. The entire training can be performed using fixed point representation or quantized training can be performed as an additional fine-tuning step after initial training with the floating point representation. In both the cases, the final trained model contains information for quantized inference. This information includes the fixed point bit-depth for each layers weights and activation. In most cases, the bit-depth for weights are selected based on the actual range in each layer, but the bit-depth for activations (inputs and outputs) are selected empirically based on the observation during training. During deployment on the embedded platform, the framework has to understand the information in these models and implement the corresponding quantization scheme. This method is not scalable for the embedded platform, as the deployment framework has to support models trained by multiple frameworks like Caffe, TensorFlow etc. This approach doesn't address 'transfer learning' which is gaining popularity due to quick deployment using pre-trained floating point models.

2.1.2 Quantized CNN Inference with Pre-trained model

In this method, the Pre-trained floating point model is converted to fixed point representation and the inference is performed using fixed point computation. This method does not require any re-training or fine-tuning avoiding the need for quantization support in the training frameworks. Models parameters (weights) are converted to fixed point representation as an offline step with fixed scale for entire inference process.

In both the methods the activations (inputs and outputs) are converted to lower bit-depth representation during every input inference process. For example, when 8-bit weights and 8-bit input activations are used for a convolution layer then the output accumulator can grow to 24-bit to 32-bit based on the convolution kernel and input activation.

The scale factor used converting this 32-bit accumulator to 8-bit representation for a given layer can be fixed for all the inference process or it can be dynamically selected for each inference process. The dynamic scale factor can be selected based on the actual range of accumulator (minimum and maximum value) in given layer during each input inference process. Utilizing the actual range for dynamic scale factor selection minimizes accuracy loss by reducing quantization error in each layer.

2.2 Layer Fusion

The most compute complexity in a CNN application is contributed by convolution layers and rest of layers in the network would need small compute. The data accesses from the memory for all the layers are similar. For example, a 3x3 convolution layer on 256 input channels feature generating 256 channel outputs would need 4608 Operations compute per output feature. A max pool layer with similar input and output channels would need 9 operations per output. An element-wise layer with 2 input tensors would need 1 operation per output feature. With the SIMD features available on the modern CPUs, the number of operation that can be performed on single CPU cycle is normally high. Except for the convolution layer, most of the other layer in a CNN application will be data bound layers. So it is important to fuse these layers with convolution layers when it is possible to utilize the compute capability of the device effectively. For example, max pooling will be performed on convolution layer on most network structure; so the max pooling operation can be performed after convolution operation on the internal memory. Similarly, element-wise addition (Residual connection in ResNet) also can be combined with convolution.

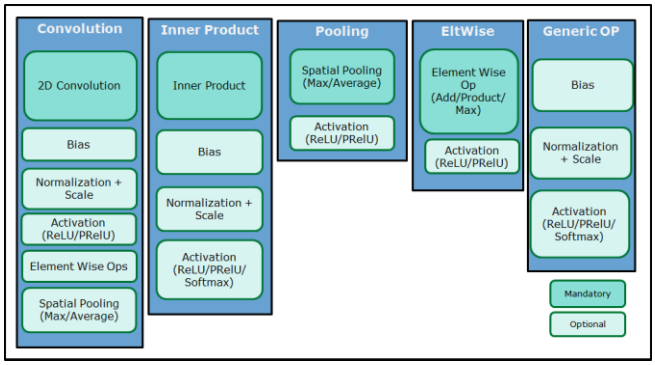

Figure 3 Vertical layer Fusion (Store Path)

Similar to an above mentioned external memory bandwidth reduction in store path (vertical fusion) as shown in Figure 3, the load path bandwidth can be reduced by processing multiple layers together when all of them use same input tensor. In case of Inception module in googleNet has multiple convolution layers and polling layers, which uses same input (horizontal Fusion). This enables a tile of input features that can be loaded once from external memory into the internal memory with in parallel reducing input data load bandwidth as shown in Figure 4.

The usage of horizontal and vertical fusion enables the external memory bandwidth reduction as well as the better resource/compute utilization.

2.3 Sparse Convolution

The complexity of the overall network should be restricted such that it fits well within the computing capability of the targeted device. Typically, the convolution layers are the most computationally intense and will determine how fast the inference runs-so it is important to reduce the complexity of convolution layers. Embedded inference can implement sparse convolution, which can execute the inference much faster when there are a lot of zero coefficients. Using sparse convolution algorithms eliminates the need for multiplications whenever the weights are zeros. Sparse training methods can induce 80 percent or more sparsity in most convolution layers - in other words, making 80 percent of the convolution weights zero. We have observed a 4 times execution speed increase when nearly 80 percent of the weights in the convolution layers are zeros. The figure 5 shows high-level control flow of convolution layer implementation to take advantage of zero coefficients in convolutions. A block multiply accumulation operation is performed on input channel if the kernel co-efficient is non zero [9].

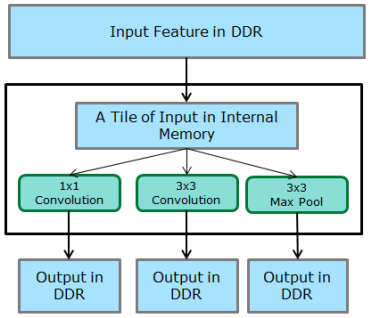

Figure 4 Horizontal layer Fusion (Load Path)

Note that the block size can be chosen appropriately to suite the local memory or data cache available in the system. In that case, the whole image will have to be split into several blocks aka ROIs (Region Of Interest).

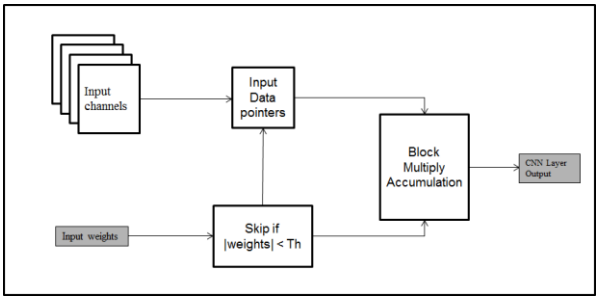

Figure 5 Sparse Convolution Control Flow

3. Proposed Free-Space Detection Application

Embedded friendly CNN network is developed for Free-Space Detection Application, which enables the car to decide the drivable path. The following sub-section explains network details and later sub-section explains optimized inference engine on TI's TDA family of automotive processor.

3.1 Network configuration

The base classification network was inspired by ResNet10 [12]. The ResNet10 network architecture was modified with the following changes.

- Residual connections are removed since it doesn't help much at small depths such as 10 as observed in the original ResNet paper[12]. Added groups of 4 to every

alternate layer to reduce complexity. Grouped convolutions also help in data bandwidth reduction.

- Max pooling is used instead of strides. This base network is used to train on the 1000 class ImageNet dataset.
- The proposed network has additional decoder layers (using deconvolution layers) on top of the base network to enable free space detection.
- The proposed network is trained for Cityscapes [13] dataset using the pre-trained weights.
- The proposed network is known as 'JSegNet21', since it has 21 convolutional and deconvolution layers. Most of the complexity of this network is concentrated in layers 13 and 14 because the Max pool stride before these layers is removed. Further details are given in Table 1.

Table 1 Layer structure of JSegNet21 segmentation network

Layer No	Layer type	Input Layer No	Output Channels	Kernel Size, Stride, Group, Dilation
1	Conv,Relu		32	5,2,1,1
2	Conv,Relu		32	3,1,4,1
3	Maxpool			2,2,-,-
4	Conv,Relu		64	3,1,1,1
5	Conv,Relu		64	3,1,4,1
6	Maxpool			2,2,-,-
7	Conv,Relu		128	3,1,1,1
8	Conv,Relu		128	3,1,4,1
9	Maxpool			2,2,-,-
10	Conv,Relu		256	3,1,1,1
11	Conv,Relu		256	3,1,4,1
12	Maxpool			**1,1,-,-**
13	**Conv,Relu**		512	3,1,1,**2**
14	**Conv,Relu**		512	3,1,4,**2**
15	Conv,Relu	14	64	3,1,2,4
16	Deconv		64	4,**2**,64,-
17	Conv,Relu	8	64	3,1,2,1
18	Eltwise	16,17		
19	Conv,Relu		64	3,1,1,1
20	Conv,Relu		64	3,1,1,4
21	Conv,Relu		64	3,1,1,4
22	Conv,Relu		64	3,1,1,4
23	Conv,Relu		8	3,1,1,1
24	Deconv		8	4,**2**,8,-
25	Deconv		8	4,**2**,8,-
26	Deconv		8	4,**2**,8,-
27	Argmax			

3.2 TI Deep learning Framework

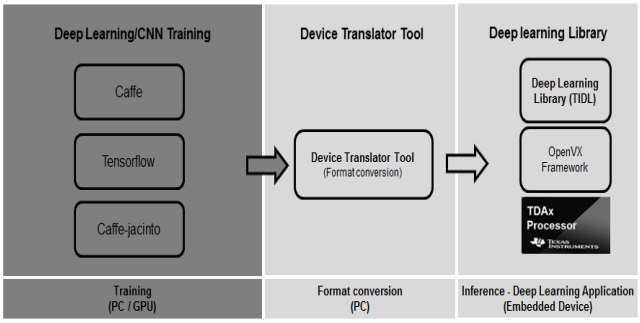

Figure 6 TI Deep learning Library suit

TI's Deep Learning library (TIDL) framework is optimized inference engine to accelerate CNN inference on TI TDA devices as shown in figure 6. TIDL does not address the training of deep-learning models, which the popular deep learning frameworks can best handle. Instead, TIDL addresses the inference part of deep learning, using a trained model from a supported network and running it at a very high speed on a supported low-power embedded processor like one from the TI TDA family. The TI device translator tool enables development on open frameworks and provides push-button PC-to-embedded porting. TIDL abstracts embedded development, provides high-efficiency implementation and is platform scalable. Features As we discussed, the purpose of TIDL is to enable ease of use and provide optimized inference. Ease of use is achieved by providing a way to use the trained network models in the TIDL library. Thus, one primary feature is that TIDL can understand the trained output of popular frameworks. TIDL has achieved optimized inference through software optimizations that enable it to use the underlying hardware resources optimally and through algorithmic simplifications, such as sparse convolutions, fixed-point computation, layer fusion etc. to reduce the inference time required for CNN.

4. Experiments and Results

The proposed embedded techniques were implemented on the TDA2x SoC from Texas instruments [14]. It is a low power SoC that operates in single digit Watts of power. It has 4 Embedded Vision Engines (EVEs), which are co-processors suited for computer vision applications.

4.1 Accuracy benchmarking

Table 2 summarizes change in accuracy due to sparsification and quantization techniques discussed in section 2. It is seen that the pixel accuracy loss is almost negligible and the change in Mean IOU loss is reasonable. Sample images are shown indicating example segmentations in Figure 7.

Table 2 Impact of sparsification and quantization, for semantic segmentation on the Cityscapes dataset

Configuration	80% sparsity induction	
	Pixel Accuracy (%)	Mean IOU (%)
Initial L2 regularized training	96.20	83.23
L1 regularized fine tuning	96.32	83.94
Sparse, fine tuned	**96.10**	**82.86**
Sparse, Quantized (8-bit dynamic fixed point)	95.90	82.15
Overall impact due to sparsification and quantization	**-0.42**	**-1.79**

Figure 7 Sample input images and the segmentation produced using sparse (80%), quantize

4.2 Inference latency on device

Table 3 compares the complexity of the original and its corresponding sparse form for 'JSegNet21' CNN model. It captures the actual Giga MACS and Giga Cycles measurements from TDA2x SoC for inferring the free space detection of one frame of size 1024x512 and number of. frames achieved per Second. Without utilizing sparsity the FPS that can be achieved is 5.14. As shown in table 3, the performance speed up is factor of 3.93x by using sparse convolution.

Table 3 Measurements from TDA2x SoC for inferring semantic segmentation of an image of 1024x512 resolution

Inference method	Configuration for inference	Giga Macs	Giga Cycles	Time (Milli-Seconds)	Frames Per Second
Dense	Without utilizing sparsity	8.843	0.700	194.44	5.14
Sparse	L2 regularized trained	8.163	0.653	181.39	5.51
	L1 regularized trained	3.264	0.299	83.06	12.04
	Sparsity induced at 80%	1.540	0.188	52.22	20.22

5. Conclusion

Mass deployment of self-driving technology requires a CNN solution, which is low cost and power optimized inference on the embedded platform. The paper introduces multiple embedded techniques namely fixed-point quantization, network optimization, and sparse convolution. These embedded techniques were implemented in TI's Deep learning library for CNN acceleration on TI's TDAx family of automotive processors. These techniques enable the real-time demonstration of free space detection with improvement in the inference performance by factor of four with minimal accuracy loss.

6. References

[1] Society of Automotive Engineers. https://www.sae.org/

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems, pages 1097–1105, 2012)

[3] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556,2014

[4] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed,D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 1–9, 2015.

[5] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, Dmitry Kalenichenko. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. arXiv:1712.05877, 2017

[6] Philipp Gysel, Mohammad Motamedi & Soheil Ghiasi. Hardware-oriented approximation of convolutional neural networks. arXiv:1604.03168,2016

[7] Lin, D., Talathi, S., and Annapureddy, S. Fixed Point Quantization of Deep Convolutional Networks. arXiv:1511.06393, 2015.

[8] Matthieu Courbariaux & Jean-Pierre David. Training deep neural networks with low precision multiplications. ICLR 2015

[9] Manu Mathew, Kumar Desappan, Pramod Kumar Swami, Soyeb Nagori. Sparse, Quantized, Full Frame CNN for Low Power Embedded Devices. CVPR workshop paper, 2017.

[10] Pete Warden. How to Quantize Neural Networks with TensorFlow.https://petewarden.com/2016/05/03/how-to-quantize-neural-networks-with-tensorflow/ 2016.

[11] Manu Mathew, Kumar Desappan, Pramod Kumar Swami, Soyeb Nagori. Sparse, Biju Moothedath Gopinath, Embedded low-power deep learning with TIDL. http://www.ti.com/lit/wp/spry314/spry314.pdf

[12] Kaiming He Xiangyu Zhang Shaoqing Ren Jian Sun, Deep Residual Learning for Image Recognition, CVPR, 2016.

[13] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The Cityscapes Dataset for Semantic Urban Scene Understanding," in Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016.

[14] TDAx ADAS SoCs, http://www.ti.com/lsds/ti/processors/dsp/automotive_processors/tdax_adas_socs/overview.page