

Real-time Pedestrian Detection using Convolutional Neural Network on Embedded Platform

Jun Hu, Wei Liu, Shuai Cheng, Huan Tian, Huai Yuan, and Hong Zhao
Northeastern University

ABSTRACT

The convolutional neural network (CNN) has achieved extraordinary performance in image classification. However, the implementation of such architecture on embedded platforms is a big challenge task due to the computing resource constraint issue. This paper concentrates on optimization of CNN on embedded platforms with a case study of pedestrian detection in ADAS. The main contribution of this proposed CNN is its ability to run pedestrian classification task in real time with high accuracy based on a platform with ARM embedded. The CNN model has been trained with GPU locally and then transformed into an efficient implementation on embedded platforms. The efficient implementation uses dramatically small network scale and a lightweight CNN is obtained. Specifically, parameters of the network are compressed by adopting integer weights to reduce computational complexity. Meanwhile, other optimizations have also been proposed to adapt the general ARM processor architecture. Finally, a robust and efficient CNN architecture is executed to run pedestrian detection algorithm in real time. The embedded platform employs a 1GHz Cortex-A53 ARMv8 based CPU. The network performs nearly 200 regions of interest detection per second with 97% accuracy by using a single core of Cortex-A53 in real traffic scene.

CITATION: Hu, J., Liu, W., Cheng, S., Tian, H. et al., "Real-time Pedestrian Detection using Convolutional Neural Network on Embedded Platform," *SAE Int. J. Passeng. Cars – Electron. Electr. Syst.* 10(1):2017, doi:10.4271/2016-01-1877.

1. INTRODUCTION

The convolutional neural network(CNN) has become ubiquitous in computer vision tasks such as image classification [1], object detection [2,3,4,5] and semantic segmentation [6,7,8]. Pedestrian detection is a popular research topic due to its paramount importance for a number of applications, especially in the fields of ADAS and automotive industry. In the context of pedestrian detection [9,10,11,12], there has been a surge of interest in CNN during the last few years.

Such approach yields good results in terms of detection accuracy and these large neural networks are very powerful. But their size consumes considerable storage, memory bandwidth, and computational resources, making them difficult to deploy on portable devices and embedded systems of the ADAS with limited hardware resources. Thus, it is important to accelerate the test-time performance of CNN.

There have been a series of studies on accelerating CNNs, such as decomposing layers [13,14,15], FFT-based algorithms [16,18], prune connections [17] and hardware-based optimizations [19]. Such ¹ methods speed up the time to some extent at no cost in accuracy or a graceful accuracy degradation based on large scale CNNs.

Though the accelerating method improves the performance, CNN with tens of millions of parameters is not suitable for the embedded systems of the ADAS with limited hardware resources and requires real-time processing. Our goal in designing a light CNN and compressing the parameters of CNN is to reduce the hardware required to run such networks, so they can be run in real time on embedded devices of ADAS by the hardware speedup techniques for pedestrian detection.

The paper proposes several novel contributions. We present a two-stage pipeline to reduce the storage and computation required by neural network in a manner that preserves the accuracy. A lightweight CNN (L-CNN) has been designed to reduce the parameters of the network. Then, the weights are quantized so that only the effective weights need to be stored. Finally, we apply the optimization techniques to enhance the performance of CNN-based general ARM processor architecture.

The rest of the paper is organized as follows: Section 2 presents the related works about detecting pedestrians and accelerating the time computation of CNN, whereas Section 3 reports all the proposed optimizations that improve the performance of the network and the platform in real traffic scene. Section 4 is devoted to experimental evaluation, whereas conclusions are drawn in Section 5.

1. This work was supported in part by the Fundamental Research Funds for the Central Universities of China under Grant N151802001.

2. RELATED WORK

Pedestrian detection methods have employed CNN showing impressive accuracy gains. LeCun et al. [9] are the first to use CNN for detecting pedestrians and propose an unsupervised deep learning approach. Hosang et al. [12] propose a supervised deep learning approach for image classification in order to detect pedestrians. Ouyang et al. [10] introduce a part deformation layer into CNN models to infer part visibility.

However, CNN models are known to be very slow especially as sliding-window classifiers. It is thus of practical importance to accelerate time performance of CNNs.

Decomposing convolutional layers [15] for accelerating time computation of CNNs is composed of two components: (i) a layer decomposition design, for example, low-rank decompositions and CP-decomposition and (ii) an optimization scheme, such as conjugate gradient descent and SGD. Despite some promising preliminary results, the whole-model acceleration of very deep networks is still an open problem. Besides the research on decomposing layers, there have been other streams on improving performance of CNNs. FFT-based algorithms are applicable and particularly effective for large spatial kernels.

Pruning redundant connections [17] have been presented to improve the energy efficiency and storage of neural networks without affecting accuracy by finding the right connections.

Compared with the improvement based on software, Vanhoucke et al. [19] represent the techniques that can be used to reduce this computational cost considerably on modern x86 CPUs. Clement Farabet et al. [20] propose a scalable hardware architecture to implement large-scale CNNs based on FPGA implementation.

3. LIGHTWEIGHT CONVOLUTIONAL NEURAL NETWORK

In this section we describe how we implemented a lightweight CNN and tested the network on embedded platforms. Section 3.1 describes how to achieve the balance between the computing complexity and accuracy of a CNN. Section 3.2 describes the skill we used to boost the accuracy and the ability of generalization. Section 3.3 describes how to transform the weights of the CNN from float to integer to reduce the complexity of the convolution computing especially on embedded platforms. Section 3.4 describes the technology to enhance the performance on embedded platforms based on ARMv8 CPU.

3.1. Lightweight Design

The convolutional operations which combined with lots of multiplication and addition operations in CNN are the most important but very expensive especially on embedded platforms. Our goal is to train a lightweight CNN and optimize the performance of the convolutional operations with acceptable accuracy that can be implemented on embedded platforms. We now present two methods of reducing the computing complexity of the networks.

Computing Resource Consumption Definition

Figure 1. shows a typical CNN architecture combined with convolutional layers and full connected layers. Time complexity [21] of convolutional layers can be defined as:

$$O\left(\sum_{l=1}^d n_{l-1} \cdot s_l^2 \cdot n_l \cdot m_l^2\right) \quad (1)$$

Here l is the index of a convolutional layer, and d is the depth (number of convolutional layers). n_l is the number of filters (also known as “width”) in the l_{th} layer. n_{l-1} is also known as the number of input channels of the l_{th} layer. s_l is the spatial size (length) of the filter. m_l is the spatial size of the output feature map.

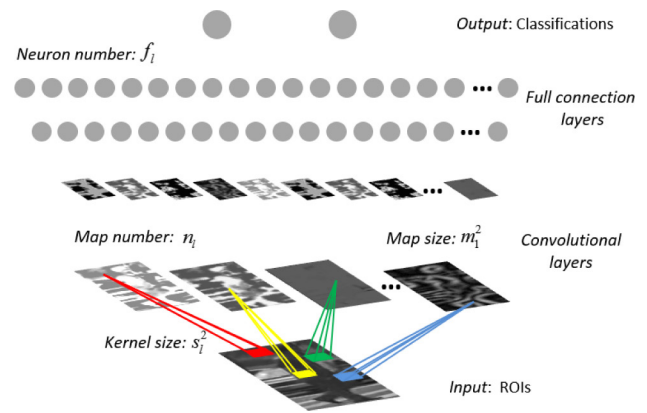


Figure 1. Example of a typical convolution neural network architecture.

Time complexity of fully connected layers can also be defined. If $l - 1_{th}$ layer is a convolutional layer, the time complexity of the first fully connected layer can be defined as:

$$O(n_d \cdot s_d^2 \cdot f_1) \quad (2)$$

Here n_d is the number of filters in the last convolutional layer. s_d is the spatial size of the filter. f_1 is the number of neurons in the first fully connected layer.

The time complexity of the other fully connected layer can be defined as:

$$O\left(\sum_{y=2}^e f_{y-1} \cdot f_y\right) \quad (3)$$

Here y is the index of a fully connected layer, and e is the depth. f_l is the number of neurons in the fully connected layer.

Furthermore, we should exactly quantify the computing resource used by a CNN. The quantity of multiplication operations can be calculated:

$$Q_{multi} = \sum_{l=1}^d n_{l-1} \cdot s_l^2 \cdot n_l \cdot m_l^2 + n_d \cdot s_d^2 \cdot f_1 + \sum_{y=2}^e f_{y-1} \cdot f_y$$