Implementation of Deep Convolutional Neural Net on a Digital Signal Processor

Elaina Chai

December 12, 2014

1. Abstract

In this paper I will discuss the feasibility of an implementation of an algorithm containing a Deep Convolutional Neural Network for feature extraction, and softmax regression for feature classification, for the purpose of real-time lane detection on an embedded platform containing a multi-core Digital Signal Processor (DSP). I will explore the merits of using fixed point and floating point arithmetic in the implementation, and provide calculations estimating whether the DSP is capable of real-time operation. While initial calculations suggest that the DSP is capable of real-time operation using floating point and fixed point arithmetic ($\sim$50Hz and $\sim$60 Hz respectively), problems were encountered using 16-bit fixed point in Q-format, resulting in a failure of the algorithm to properly classify lanes. Future work for this project include exploring the challenges associated with the fixed point implementation, as well completing the migration of the algorithm to multi-core DSP using BLAS libraries.

2. Introduction

In recent years, deep convolutional neural networks, have gained large amounts of interest in the machine learning community due to their high performance in object recognition challenges such as MNIST and ImageNet ([1]). Applications for Deep CNN are constantly growing, from object recognition, to speech recognition and more recently lane detection.

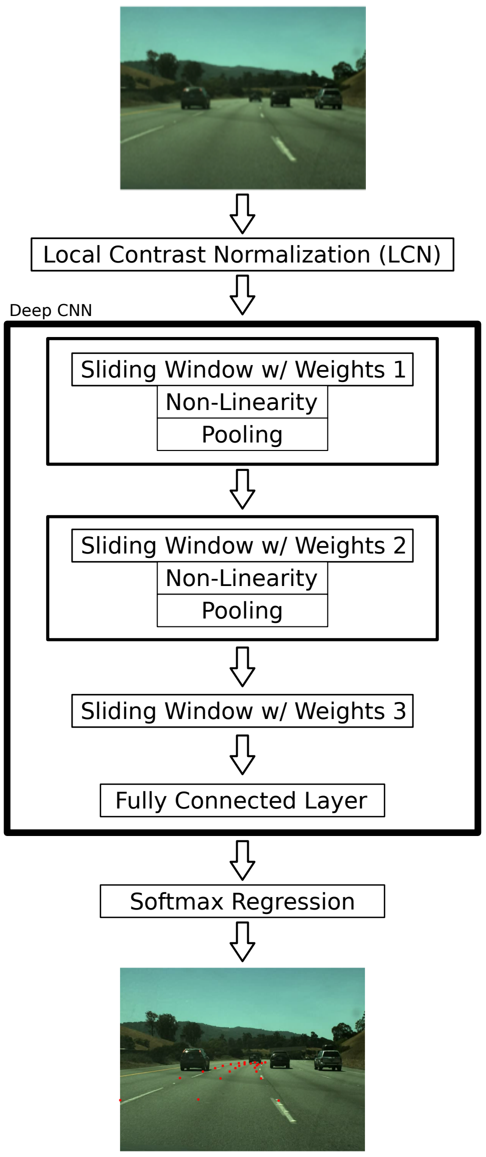
Although there are many applications which can benefit from machine learning (or other statistical learning algorithms), current machine learning implementations are typically run on expensive high-powered GPU's. These implementations are infeasible for applications that require relatively inexpensive hardware, or more importantly, are severely energy constrained.

DSP's, where performance/Watt benchmarks are of particular interest, are well suited for any application requiring large amounts of computation under strict energy constraints. However, many of these energy gains are achieved by doing 16-bit fixed point calculations, as opposed to single precision floating point typical of CPU and GPU architectures. While a move to 16-bit fixed point would result in precision loss in the calculations, this may not be a problem in machine learning applications. In the inference stage of machine learning algorithms, such as softmax regression, final classification is usually done by comparing calculated probabilities, as opposed to making decisions on based on absolute values. As a result, machine learning algorithms, which already have some amount of tolerance to noise, also have some amount of tolerance to precision loss. Some questions that I will explore are the following:

- What levels of precision loss (due to truncation from the migration from floating point to fixed point, with a limited word and fractional length) can be tolerated before noticeable decreases in accuracy are observed?
- What is the theoretical frame rate that can be achieved using embedded hardware, using floating or fixed point arithmetic?

To explore these questions, I implemented a Lane Detector Algorithm using a Deep Convolutional Neural Network for feature extraction, and a Softmax Regression Algorithm for feature classification using C++, with end goal being a real-time implementation on an embedded Digital Signal Processor (DSP) platform. I created fixed and floating point versions to better understand effect of precision loss on the algorithm accuracy. I also performed initial calculations to better understand the theoretical limits of the hardware platform.

3. Lane Detector Algorithm


Local Contrast Normalization (LCN)
Deep CNN
Sliding Window w/ Weights 1
Non-Linearity
Pooling
Sliding Window w/ Weights 2
Non-Linearity
Pooling
Sliding Window w/ Weights 3
Fully Connected Layer
Softmax Regression

4. Description of Project Development Platform

The hardware used in this project is the 66AK2H Evaluation Model (EVM) by Texas Instruments (TI). The EVM is a development platform containing a single 66AK2H Keystone Multicore DSP+ARM System-on-Chip (SoC) by TI. The platform contains 10GB Ethernet interfaces as well JTAG Emulation and RS-232 Serial Com Interfaces to facilitate communication to the on board SoC. The SoC contains 4 ARM A15 cores, and 8 C66x DSP Cores, with the following key features:

- 38.4 GMacs/DSP Core for Fixed Point @1.2 GHz
- 19.2 GFlops for Floating Point @1.2 Ghz

Initial code development for this project is Xcode 6.1 using C++. Code was initially tested on an Intel i7 CPU, and cross compiled to run on the ARM+DSP Core using the Linaro Linux GCC 4.7 Cross Compiler. The final implementation will use BLAS libraries and compilers provided in the TI MCSDK-HPC development tools.

5. Theoretical limits of DSP Hardware with respect to Algorithm

In this section I discuss the theoretical limits of the hardware platform. I explore whether the EVM has the capability to store and compute the lane detector algorithm in real-time

5.a. Memory Capacity Requirements

To realize 20 Hz "real time" operation, ideally, the on-board, i.e. EVM memory must have the capacity to hold the following:

- All coefficient parameter data ∼160MB
- Image Data ∼ 4MB/image
- program data ∼ 0.14 MB

To maximize computation throughput, the EVM must have the capacity to hold all the coefficients, as well as at least 1 image . This means that we need $160MB + 4MB + 0.14MB \sim 165MB$ of memory on the EVM.

6. Calculating Computation time of a 2D Convolution

A calculation of the total amount of operations required by the algorithm shows that the computation is entirely dominated by the sliding window convolutions, but at least two orders of magnitude. Each sliding window convolution requires $>5 * 10^8$ FLOPS per image, vs $>10^6$ FLOPS for other parts of the algorithm. The c6678 DSP used in the hardware platform are capable of 8 multiply-accumulate (MAC) operations per cycle for single precision floating point. Assuming a 1 Ghz clock, this roughly corresponds to a ∼7Hz frame rate for a single core, or a ∼50Hz frame rate using all 8 DSP cores.

Despite a theoretical maximum of 32 MACs/cycle for a 16-bit fixed point implementation, the frame rate for a 16-bit fixed point implementation may not be much higher than the floating point frame rate. This is due to data movement restrictions. For efficient movement of data through the memory, the data has to be properly aligned in memory. Additionally, highly optimized matrix-matrix functions available for the DSP have restrictions on the dimensions of the dot-product computations. The result is that the frame rate for a fixed point implementation may only be ∼60 Hz if using all 8 DSP cores.

7. Current Results and Difficulties

The Lane Detector Algorithm was implemented in C++ in two forms: single-precision floating point and fixed point. The fixed point implementation uses a very coarse but flexible fixed point implementation:

- All parameters are converted from single-precision floating point to an Q-format fixed point
- All computation carried out in Q-format
- Final values converted back to floating point for comparisons against reference data

The final values of the softmax regression step were compared to a reference file generated by the original CUDA + Caffe GPU implementation of the lane-detector algorithm. Additionally, the single-precision implementation was cross-compiled and run on an a single ARM A15 core as an initial test of the hardware platform setup. The final values of these implementation were also compared to the reference output.

7.a. Summary of Results

	Max Difference	Normalized MSE
Single Precision Floating Point (C++ CPU)	8.0341e-05	3.3454e-07
Single Precision Floating Point (C++ ARM)	0.4805	4.7661e-03
16-Bit Fixed Point (MATLAB)	0.0036	6.0865e-08
16-Bit Fixed Point (C++ CPU) (Q-format)	31.62	0.7360
32-Bit Fixed Point (C++ CPU) (Q-format)	0.8759	0.0104

7.b. Fixed Point

The current 16-bit fixed point implementation fails to classify the lanes. More work will need to be done to pinpoint where the algorithm fails.

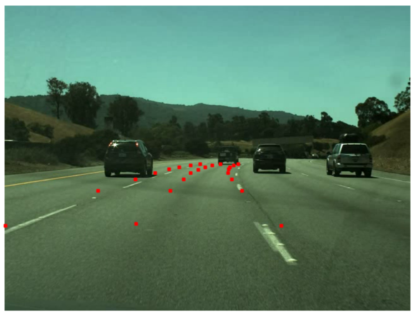


Figure 7.1: 32-bit Fixed Point Output

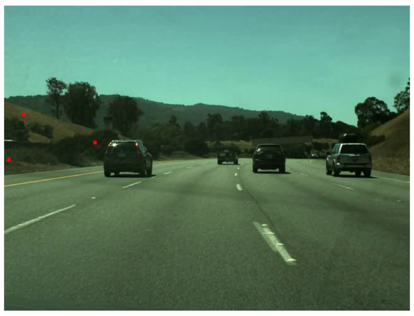


Figure 7.2: 16-bit Fixed Point Output

7.c. Precision Errors between Cross-Platform Implementations

As an initial test of the TI embedded platform, the C++ floating point implementation was cross-compiled and run on a single ARM core. While there was a much higher difference between the final values and the reference output, for the floating point implementations were able to correctly classify the lanes in the input image:

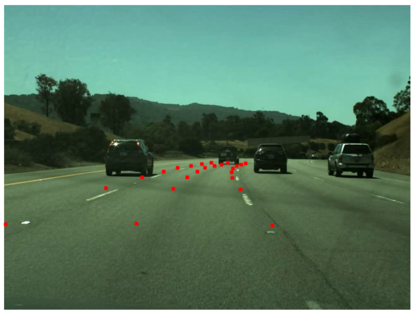


Figure 7.3: Reference Output

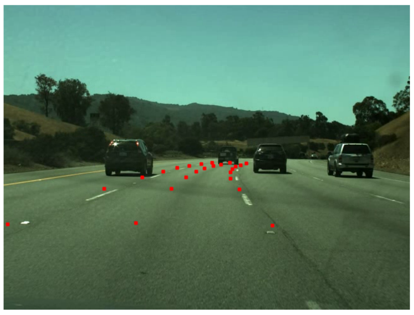


Figure 7.4: ARM Floating Point Output

8. Future Work

High Speed multi-core DSP's have the theoretical capability for real-time low power implementation of a feed-forward Deep Convolutional Neural Network at an order of magnitude less power than a conventional CPU or GPU. There are current difficulties at this time with the 16-bit fixed point implementation. Additionally, initial calculations seems to suggest that the fixed point implementation may not achieve a frame rate that much greater than the floating point implementation for current filter window sizes. As a result, while I will work to further to understand where the algorithm breaks down in the fixed point implementation, I will also focus the migrating the the floating point implementation to the 8 DSP cores.

I will focus on achieving an implementation optimized for efficient data movement through the memory. To achieve this, I will focus on a careful migration using BLAS libraries designed for parallel computation across all 8 TI DSP cores. I am currently in talks with the High Performance Computing (HPC) lab from TI on how to best do this.

9. Acknowledgements

This project could not have been done without the efforts of the following people: Charles Qi, who provided the original Matlab implementation I am basing this work on, Boris Murmann and Andrew Ng for their support of this project, Sameep Tandon and Tao Wang for their advice and for providing the image and parameter data for the algorithm, and Fernando Mujica for his advice and support as well as for providing the hardware platform.

References

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. *Advances In Neural Information Processing Systems*, pages 1–9, 2012.