Journal of Systems Architecture 61 (2015) 693–699

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

An embedded system for handwritten digit recognition

Luca B. Saldanha *, Christophe Bobda

CSCE Department, University of Arkansas, Fayetteville, AR, United States

ARTICLE INFO

Article history:
Received 10 April 2015
Accepted 28 July 2015
Available online 18 August 2015

Keywords:
Neural network
Soft processor
Regularization
Image processing

ABSTRACT

The goal of this work is the design and implementation of a low-cost system-on-FPGA for handwritten digit recognition, based on a relatively deep and wide network of perceptrons. In order to increase the performance of the application on embedded processors whose performances are way below standard general purpose CPUs, a regularization method was used during the training phase of the neural network that allows for the drastic reduction of floating point operations. Our implementation achieves a 3× speed-up toward a raw implementation without optimization, while keeping the accuracy in acceptable ranges. Our efforts reinforce the fact that FPGAs are suited for deploying complex artificial intelligence modules.

1. Introduction

Artificial intelligence is on the verge of becoming ubiquitous in our society. A considerable amount of money made by Internet companies comes from systems based on machine learning and user data mining for recommending movies, personalized advertisement and even match making. Soon autonomous intelligent systems will share our world. Some already do, especially to substitute humans on high risk activities. But these are very sophisticated and expensive systems. The barriers imposed by several constraints are still to be broken: size, weight, power consumption (SWAP) and cost, for example.

Embedded systems using reconfigurable devices appear as a possible solution. Even though the most complex problems in the artificial intelligence field still require supercomputer-like performance, FPGAs are already an alternative for less complicated tasks. This work demonstrates that by applying field specific knowledge and by exploiting the full range of capabilities that an FPGA is able to offer, it is possible to use them to implement practical systems that rely on relatively deep and wide neural networks, such as handwritten digit recognition.

In 2011, Gomperts and Ukil [1], already devised an implementation totally with VHDL of an artificial neural network. They stressed that this unconventional computational model presents complications such as a high hardware cost and a large number of arithmetic operations (especially floating or fixed point). Dinu et al. [2] presented an algorithm for hardware implementation of neural networks, but with a simplified activation function that allows for the simplification of the calculations due to its boolean properties. Another successful and low cost implementation of a learning structure (support vector machines) in FPGA is be found at [3].

All these hardware models still require a great engineering effort, in the sense that for every specific problem, the designer must be aware of a great number of details that are not always clear at a high level abstraction. This paper declaims a simple approach. Since FPGA logic is inadequate for complex floating point operations, embedded soft processors, such as the MicroBlaze [4], are targeted or, more recently introduced hard coded processing cores embedded in the FPGA fabric ([5] for example). Our work proposes to circumvent the use of floating point operations through a combination of code re-organization and dedicated hardware accelerators. The hardware blocks are used to process the input data, making useful transformations to conveniently feed the neural network. Further explanation on the system architecture is provided in Sections 2 and 5.

Crunching the input data has a clear positive effect when it comes to improving the accuracy of the system, but it does not directly augment its performance. The effect is rather incidental. As it is demonstrated in subsequent sections, excluding non-relevant sections of the image in the case of digit recognition (e.g. the background), significantly reduces the number of arithmetic operations in the neural network. This fact alone, however, is not enough. Further improvements on the training phase of the neural network shall be carried out.

An established concept in computational models claims that, with very high probability, the best model is not more complex

* Corresponding author.
E-mail address: lbochisa@uark.edu (L.B. Saldanha).
URL: http://www.csce.uark.edu/~lbochisa/ (L.B. Saldanha).

http://dx.doi.org/10.1016/j.sysarc.2015.07.015
1383-7621/

than the data suggests. This line of thought is referred to as the Occam's razor [6]. Several methods are applied to the training process of neural networks to take advantage of this property [7]. L1 and L2 regularization are an example, as demonstrated in [8,9]. They exploit the hypothesis that some neural network connections may be discarded and promote simplicity in the structure (the case of L1 regularization) or the hypothesis that weaker connections (small-scaled weights) have a similar simplifying effect (L2 regularization). In the remaining of the paper it is demonstrated that L1 regularization has an interesting side effect that may be applied to our problem. By discarding the majority of the neuron's connections, the neural network becomes feasible in very simple embedded processors, with low clock frequency and extremely low cost.

By exploiting this side effect of L1 regularization, a handwritten digit recognition system relying on a neural network predictor trained using this method could be designed and implemented. It achieves a 3× speed-up compared to the same system trained without any optimization, making it possible to be executed in a simple soft processor.

The rest of the paper is organized as follows. Section 2 gives the reader an overview of the proposed architecture. Then, Section 3 lays the mathematical ground that is fundamental for the understanding of our approach. Next, the training process is explained in Section 4 and every hardware block is briefly described (as well as the software design approach) in Section 5. Finally Section 6 shows the results and Section 7 concludes the paper as well as briefly discusses the future work.

2. System overview

The system is described overall as a hardware/software solution for the specific application of handwritten digit recognition. Hardware accelerators are used for the image processing pipeline, liberating the software from the heavy burden of sequentially dealing with image pixels. Another advantage is to significantly reduce data exchange, a frequent bottleneck for System-on-Chips (SoCs) performance, with external memories. Fig. 1 depicts a high level schematic of the whole system. Every block was written in VHDL, except for the MicroBlaze processor and DDR3 controller, which were generated using tools from the FPGA manufacturer.

The input data must be processed properly in order to comply with existing datasets used for off-line training (see Section 4). There is nothing special in our implementation for the image filters, except, perhaps, for the proper choice of operations and their order. The image is captured with a regular digital camera and the blocks *Camera Interface* and *Bayer to Grayscale* treat the input pixels to the desired grayscale format. Then, a thresholding operation takes place to discard the irrelevant background and random noise that might have occurred. Finally, a series of Gaussian filters is applied, followed by subsampling, in order to reduce the size of the image to the desired amount of pixels. All relevant details of these hardware blocks are described in Section 5.1.

Hardware blocks for convolution and thresholding operations usually exhibit real time behavior while keeping the resource usage to a minimum. On the other hand, floating point operations using dedicated DSP blocks (e.g. Xilinx's DSP48A1) are commonly not directly supported. Even custom made blocks tailored specifically for a given task require a significant portion of FPGA logic. Therefore, implementations of neural network based predictors, with a high number of neurons, are not suited for parallel processing, making a sequential processor with floating point operations capabilities fundamental. The *MicroBlaze* processor is used to deploy the neural network based predictor, its output is used by the *Drawing Module* to render the predicted digit onto the image (for debugging purposes) and finally, after passing through a DDR3 based frame buffer, the result is shown in a TFT display.

This is a typical example of the importance of hardware and software mixed solutions. The central task for the application can not be implemented in hardware blocks alone (since the device does not have enough resources) and it is too slow to execute purely as software in the low-performance embedded processors. The former obstacle can not be overcome, at least without a great engineering effort, but the second may be tackled using software optimization methods and by understanding the elemental properties of the problem at hand. The next section goes deep into the subject and a possible solution is demonstrated.

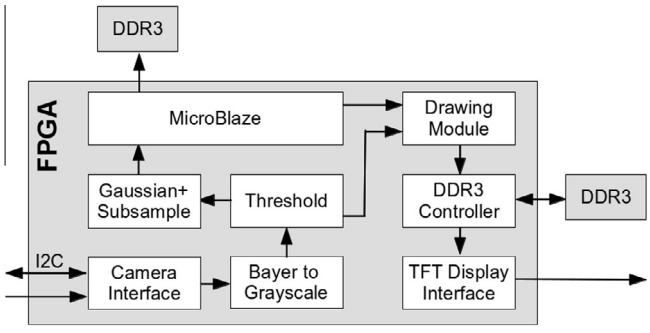

Fig. 1. System overview.

3. Neural networks

The implementation of a relatively complex neural network on a cheap, small and low-power FPGA using a standard soft-processor as the main processing unit is the fundamental contribution of this work. Thus, it is mandatory to deepen our knowledge in this matter. This section briefly guides the reader through the basics of neural networks based on standard, non-linear, continuous activation functions. Then, the text goes over the training process based on gradient descent and backpropagation. Finally, it is shown that by using an L1 regularization method the number of weights of the neural network may be drastically reduced, making it possible to be implemented in a low-frequency embedded processor in real time.

3.1. Basics

Neural networks have been studied for a long time. They date back to at least 1943 [10] but the principles involved may be traced to early works on linear regression methods in the 1800s as noted by [11]. A great number of neural network models was developed ever since, therefore it is important to emphasize that this work is particularly interested in multi layer perceptrons (MLP) [12]. The term *neural network* refers solely to *multi layer perceptron* throughout the remainder of the text.

A perceptron is basically a biologically inspired model of a neuron. It is excited by other neurons (or by external signals) and triggers an output based on the sum of its inputs after being scaled by weights that model how strong two perceptrons are connected. This transformation is called the activation function. Fig. 2

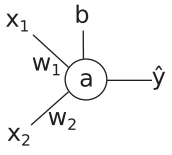

Fig. 2. The perceptron model.