Incremental Learning on Chip
Ghouthi Boukli Hacene, Vincent Gripon, Nicolas Farrugia, Matthieu Arzel,
Michel Jezequel

To cite this version:

HAL Id: hal-01754847
https://hal.archives-ouvertes.fr/hal-01754847
Submitted on 31 May 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Abstract—Learning on chip (LOC) is a challenging problem, which allows an embedded system to learn a model and use it to process and classify unknown data, adapting to new observations or classes. Incremental learning of chip (ILOC) is more challenging. ILOC needs intensive computational power to train the model and adapt it when new data are observed, leading to a very difficult hardware implementation. We address this issue by introducing a method based on the combination of a pre-trained Convolutional Neural Network (CNN) and majority vote, using Product Quantizing (PQ) as a bridge between them. We detail a hardware implementation of the proposed method validated on an FPGA target, with substantial processing acceleration with few hardware resources.

Transfer Learning, Incremental Learning, Learning on Chip, Convolutional Neural Network, FPGA

I. INTRODUCTION

Recently, Deep Neural Networks (DNNs) achieved significant progress and became the state-of-art in the field of machine learning. In particular, Convolutional Neural Networks (CNNs) now exhibit state-of-the-art performance in object recognition. DNNs rely on hundreds of millions of parameters that are trained using a large amount of data, requiring heavy computational power and memory resources. Such resources are not readily available on embedded systems such as smartphones running on battery power.

To address these limitations, many recent studies proposed to reduce the size of DNNs using product quantization (PQ) methods, in order to quantize the DNNs weights [1][2]. Other methods proposed to binarize only DNNs weights [3], as well as activation functions [4], with the aim to reduce both DNNs size and computational complexity. These methods allow the implementation of DNNs on embedded systems such as FPGA [5][6]. However the proposed hardware implementations focus only on the inference process of pre-trained DNNs, assuming that the training process has been done previously. The main weakness of these methods is that they do not propose a way to handle LOC.

LOC allows an embedded system to train a model and use it to classify and process new data. This concept represents one of the most active area research, because of the intensive computation required during the training phases, and which cannot be handled by a small embedded system with a limited power. A more challenging problem is the incremental learning on chip (ILOC). Incremental learning methods aim to learn data sequentially, adapt the model to the new data, be able to learn new examples and classes and finally do not require access to the old data to retain and adapt the model [7]. Although models have been proposed and studied extensively during the last decades, finding a good compromise between accuracy and required resources remains challenging. Indeed, most of existing works retrain the model when receiving new data [8], [9], and reuse some prior data for the retraining process [7], [10], which is computationally expensive and does not meet the embedded systems requirements. In [11], the authors introduced a new simple incremental learning method based on transfer learning, product quantization and majority vote (cf. 1). This method adapts the model to new observed examples and classes without retrain or access to previous data and, uses much less computational power than existing counterpart and approaches state-of-art accuracy on challenging vision datasets (CIFAR10 and ImageNet). These properties agree perfectly with embedded systems requirements.

In this paper we propose a hardware architecture for an ILOC solution based on [11], with the following claims:

- It is possible to adapt the model to new data (from scratch) without retraining it,
- It uses limited computational resources,
- It is used for the training and not only for inference process.

The outline of the paper is as follows. In Section II we introduce related work. In Section III we present an overview of the proposed incremental method. Hardware architecture and implementation is introduced in Section IV Hardware results are outlined in Section V. Finally, Section VI is a conclusion.

II. RELATED WORK

Learning on chip (LOC) refers to the ability of an embedded system to learn data by itself, then process and classify new unknown data. Some previous works have proposed solutions to train a model on an FPGA, using neural networks [12]. This solutions needs to implement gradient descent which is computationally expensive and quickly becomes a problem when the network size increases. Others propose to train Support Vector Machines on FPGA [13], but still this solution requires intensive computational power and large memory usage to store all training data. The need of intensive computation and memory usage during the learning phase represent a major drawback for LOC. In the last years, the interest was focused on large database as ImageNet,and this leads to propose big DNNs to handle and be able to classify these datasets. As the implementation of big DNNs is problematic, proposed works in this context focus on inference process and assume that the training or learning process is done on an external server [5][6], or use SVMs already trained on a large dataset as classifiers [14].

Incremental learning process the learning data sequentially and being able to handle new data and new classes without the need to retrain the whole system [15]. Most of existing works
either add new classifiers to accommodate new data, such as the learn++ method [7], [10] or retrain the model using newly received data together with the old model ([18], [9]). To avoid training a large number of classifiers, and to address the catastrophic forgetting problem ([16], [17]), a combination between SVMs and learn++ method called “SVMlearn++” ([18]) was proposed, showing promising improvements ([19]). However, this method still needs to retrain a new SVM each time new data is provided, and some knowledge is forgotten while new information is being learned. These methods need Intensive computing for training and large memory usage, which do not satisfy the embedded systems criterion.

In [11], the authors introduced an incremental learning model, in which the learning process consists in making a random sampling over input vectors, and then splits the obtained vectors and stores them. We describe this method in the following Section. In this paper, we will exploit the simplicity of the learning process of this method to overcome LOC problems and propose an ILOC solution.

III. OVERVIEW OF THE INCREMENTAL METHOD

The incremental method introduced in [11] relies on the use of a pre-trained deep CNN as feature extractor, followed by product random sampling to embed data in a finite alphabet. The last step of the method consists in a majority vote. We detail this method in the next paragraphs.

First, we use the internal layers of a pre-trained CNN [20], that transforms an input signal $s^m$ into a feature vector $x^m$ (cf. Figure 1 step 1). Next, each obtained feature vector $x^m$ is embedded in a finite alphabet using a PQ technique [21]. We selected product random sampling as it achieves good performance, yet it is computationally much lighter than other PQ techniques such as K-means 1. Product random sampling splits each feature vector $x^m$ into $P$ subvectors of equal size denoted $(x^m_p)_{1 \leq p \leq P}$, which are quantized independently using the $K$ anchor points $Y_p = y_{p1}, \ldots, y_{pK}$, where for each $y^m_p$, $\exists x^m \in X, x^m_p = y^m_p$.

After transforming each feature vector $x^m$ into a word of fixed length $(q^m_p)_{1 \leq p \leq P}$ (using the alphabet of anchor points)(cf. Figure 1 step 2), we associate the corresponding output $c^m$ (an indicator vector of the class) to the obtained points (cf. Figure 1 step 3). The same process is done for each new data to handle incremental learning. The combination of the pre-trained CNN as feature extractor and the majority vote as classifier allows example and class incremental learning without damage previous learned knowledge [22] or retrain the model. These properties make of the proposed method a perfect approach to be implemented on embedded system.

The method was tested on challenging vision datasets (CIFAR10 and ImageNet), using Inception V3 [23] as feature extractor. The test gave promising results, 82% of accuracy for CIFAR10 and 8% on ten categories of ImageNet distinct from the 1000 ones that were used to train the CNN.

IV. HARDWARE IMPLEMENTATION

In this paper, we present a hardware implementation for the step 2 and 3, and we assume that the step 1 (feature extractor) is done by an external CPU which will give the feature vector $x^m$ to the FPGA.

A. Data Quantization

In addition to the quantization of the feature vectors $X$, we quantize the feature vector’s values using a signed fixed point representation on $n$ bits with $5$ bits for the integral part. The main motivation of this step is to reduce the number of bits to represent a value from $32$ to $n$, where $n$ should be lower than $18$ in order to use only one DSP for each operation indeed of 2.

B. Architecture

The proposed architecture handles both the learning and classification processes. The learning process is quite simple, as described in [11]. After the random sampling, learning requires splitting the feature vectors $x^m$ into $P$ parts, then storing each part $x^m_p$ into RAMD$_P$, which represents anchor vectors denoted $y^m_p$, and the input class vector $c^m$ into RAMC$_P$ ($1 \leq p \leq P$)(cf. Figure IV-B). The input class vector $c^m$ containing the class of the feature vector $x^m$ is one hot encoded. We choose the one hot encoding to simplify the classification process described in the following paragraphs.

To classify an unlabelled feature vector $x^m$, we split the vector into $P$ parts and obtain the $P$ associated subvectors $x^m_p$, $1 \leq p \leq P$. The hardware architecture used to classify the unlabelled $x^m$ is divided into two parts, the processing and the classification part (cf. Figure IV-B). The processing hardware architecture is made of $P$ identical parts. For each $p$ part, we first compute the euclidean distance between $x^m_p$ and $y^m_p$, and store the distance in the register $r_p$. We do the same process with each $(y^k_p)_{1 \leq k \leq K}$, we compare the obtained result with the distance stored in the register $r_p$. We store the smallest distance and the vector class $c^p$ one hot coded on $C$ bits, when $C$ is the number of classes, and corresponding to $y^k_p$ which is the nearest from $x^m_p$ and gives the smallest distance. Done sequentially on all $(y^k_p)_{1 \leq k \leq K}$, this process needs $K$ clock cycle, one clock cycle to compute one distance for each $y^m_p$. The same process is running on the $P$ parts in parallel.

The classification hardware architecture takes as input the $(c^p)_{1 \leq p \leq P}$ stored in $(r_p)_{1 \leq p \leq P}$. As a first step, a bitwise addition is computed over all vectors $c^p$. The $C$ results of the additions are stored into $C$ registers and then compared. The comparison is done sequentially in such a way that we compare only two results, store the highest one and its index $c$ ($1 \leq c \leq C$), then we compare the third result with the one stored in the register, keep the highest one and its index $c$, and so on. In the end of this process, the index $c$ stored in the register presents the class that the model has attributed to the unlabelled feature vector $x^m$.

The architecture is fully pipelined, so the number of clock cycles to process an input data is the number needed to
Step 2 and the maximum frequency is 209 MHz. The energy consumption of the whole system is about 2048 Watt.

We use inception V3 [23] which gives a 16 dimensional feature vector size (cf. Table I). For the classification process, we obtain a result each K clock cycles. The experience parameters used are the same as defined in [11] to get an accuracy of 82% and 81.6% for 32 and 16 value encoding respectively. To obtain feature vectors we use inception V3 [23] which gives a 2048 dimensional feature vector.

The energy consumption of the whole system is about 2048 Watt and the maximum frequency is 209 MHz. For K = 200 the time needed to classify an input vector is 957 ns, and the ratio between a software simulation delay using an I7 870 (2.93 GHz) processor and the FPGA when P = 64 and K = 200 is $10^4$. The Table I shows a summary of the resource allocation of the FPGA for the implementation of ILOC architecture.

VI. CONCLUSION

We proposed a hardware architecture using limited resources for an incremental learning on chip, able to train a model incrementally from scratch on chip. The hardware implementation is fully parallel and pipelined. This architecture can be used in a variety of classification applications, and can be embedded inside a processor chip. The proposed architecture allows an embedded system to be trained and tested on new data, to be dynamic and easily adaptable to new changes. Future works will focus on proposing a hardware implementation for the deep CNN which acts as feature extractor to have a complete dynamic system.

REFERENCES


Figure 2. Hardware architecture of ILOC