Incremental Learning on Chip

Ghouthi Boukli Hacene, Vincent Gripon, Nicolas Farrugia, Matthieu Arzel, Michel Jezequel

Incremental learning on chip

BOUKLI HACENE Ghouthi[1,2], GRIPON Vincent[1,2], FARRUGIA Nicolas[1,2], ARZEL Matthieu[1,2], JEZEQUEL Michel[1,2]
1 : ELEC - Dépt. Electronique (Institut Mines-Télécom-IMT Atlantique-UBL)
2 : Lab-STICC - Laboratoire en sciences et technologies de l'information, de la communication et de la connaissance (UMR 6285 - CNRS - IMT Atlantique - Université de Bretagne Occidentale - Université de Bretagne Sud - ENSTA Bretagne - Ecole Nationale d'ingénieurs de Brest)

***Abstract*—Learning on chip (LOC) is a challenging problem, which allows an embeded 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 adress 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 embeded 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 autors 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 conterpart and approaches state-of-art accuracy on challenging vision datasets (CIFAR10 and ImageNet). these properties agrees 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 it self, 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 ([8], [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 $\mathbf{s}^m$ into a feature vector $\mathbf{x}^m$ (cf. Figure 1 step 1). Next, each obtained feature vector $\mathbf{x}^m$ 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 $\mathbf{x}^m$ into P subvectors of equal size denoted $\left(\mathbf{x}_p^m\right)_{1\leq p\leq P}$, which are quantized independently using the K anchor points $Y_p = \mathbf{y}_{p1}, ..., \mathbf{y}_{pK}$, where for each $\mathbf{y}_p^k$, $\exists \mathbf{x}^m \in X, \mathbf{x}_p^m = \mathbf{y}_p^k$.

After transforming each feature vector $\mathbf{x}^m$ into a word of fixed length $\left(\mathbf{q}_p^m\right)_{1\leq p\leq P}$ (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 $\mathbf{x}^m$ 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 feacture vectors $\mathbf{x}^m$ into P parts, then storing each part $\mathbf{x}_p^m$ into $RAMD_p$, which represents anchor vectors denoted $\mathbf{y}_p^m$, and the input class vector $\mathbf{c}^m$ into $RAMC_p$ $(1 \leq p \leq P)$(cf. Figure IV-B). The input class vector $\mathbf{c}^m$ containing the class of the feature vector $\mathbf{x}^m$ 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 $\mathbf{x}^m$, we split the vector into P parts and obtain the P associated subvectors $\mathbf{x}_p^m$, $1 \leq p \leq P$. The hardware architecture used to classify the unlabelled $\mathbf{x}^m$ 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 $\mathbf{x}_p^m$ and $\mathbf{y}_p^1$, and store the distance in the register r_p. We do the same process with each $\left(\mathbf{y}_p^k\right)_{1\leq k\leq K}$, we compare the obtained result with the distance stored in the register r_p. We store the smallest distance and the vector class $\mathbf{c}^p$ one hot coded on C bits, when C is the number of classes, and corresponding to $\mathbf{y}_p^k$ which is the nearest from $\mathbf{x}_p^m$ and gives the smallest distance. Done sequentially on all $\left(\mathbf{y}_p^k\right)_{1\leq k\leq K}$, this process needs K clock cycle, one clock cycle to compute one distance for each $\mathbf{y}_p$. The same process is running on the P parts in parallel.

The classification hardware architecture takes as input the $(\mathbf{c}^p)_{1\leq p\leq P}$ stored in $(\mathbf{r}_p)_{1\leq p\leq P}$. As a first step, a bitwise addition is computed over all vectors $\mathbf{c}^p$. The C results of the additions are stored into C registers and then compared. The comparison is done sequentially in a such 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 one. 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 $\mathbf{x}^m$.

The architecture is fully pipelined, so the number of clock cycles to process an input data is the number needed to

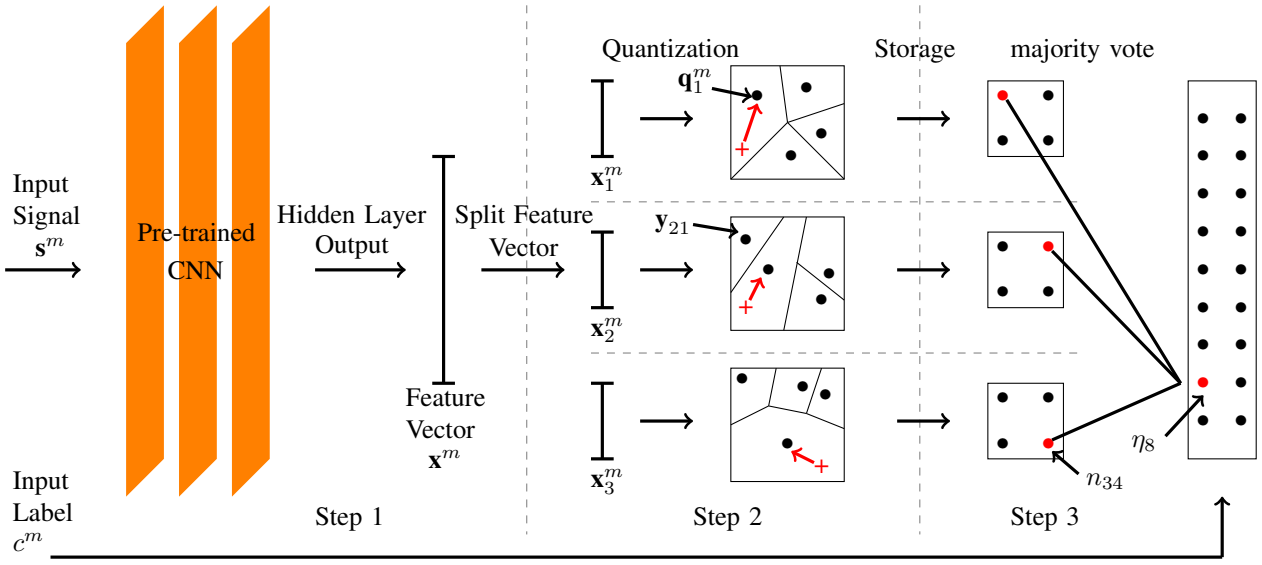

Figure 1. Overview of the proposed method, comprising three main steps. Given a set of samples, we first use a pre-trained CNN for feature extraction (Step 1). Subsequently, we use a PQ technique to quantize the feature vectors (Step 2). Finally, we use a majority vote to classify the quantized data (Step 3)

compute distances with all anchor vectors $\mathbf{y}_p^k$. This means that the model gives a valid output every K clock cycles.

V. Results

The proposed architecture has been implemented on Xilinx Virtex 7 (xc7vx690tffg1157) Field Programmable Gate Array (FPGA). Its functionality has been verified by comparing the obtained misclassified feature vectors of the hardware architecture and the software simulation. For the hardware solution, we encoded the input feature vectors on 16 bits in order to use only one DSP for each multiplication operation. This encoding reduces accuracy from 82% on CIFRA10 to 81.6%. Unlike other learning methods, this proposed approach only needs to split input vector into P parts and then store them. This allows a light learning process in which each vector is stored in one clock cycle, no operation is performed and $T \cdot K \cdot 16$ bits memory usage is used, where T is the 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. 2048 DSP are used, one for each vector value due to the parallezation of processes applied to process vector's values. The energy consumption of the whole system is about 22 Watt and the maximum frequency is 209 MHZ. for $K = 200$ the time needed to classify an input vector is 957 ns, and the ration 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.

Table I
FPGA results for the ILOC architecture ($T = 2048$, $P = 64$, $K = 200$ and $n = 16$ bits).

Memory usage dedicated to store X (bits)	6553600
Combinational Look-up Tables (LUT)	$265680/433200(61\%)$
Combinational Look-up Tables (DSP)	$2048/3600(57\%)$
Maximum frequency (MHz)	209
learning/classifying delay (per feature vector)	$4.79ns/957ns$
Software to hardware delay ratio (delay)	10^4

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

[1] Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng, "Quantized convolutional neural networks for mobile devices," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 4820–4828.

[2] Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev, "Compressing deep convolutional networks using vector quantization," *arXiv preprint arXiv:1412.6115*, 2014.

[3] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Advances in Neural Information Processing Systems*, 2015, pp. 3123–3131.

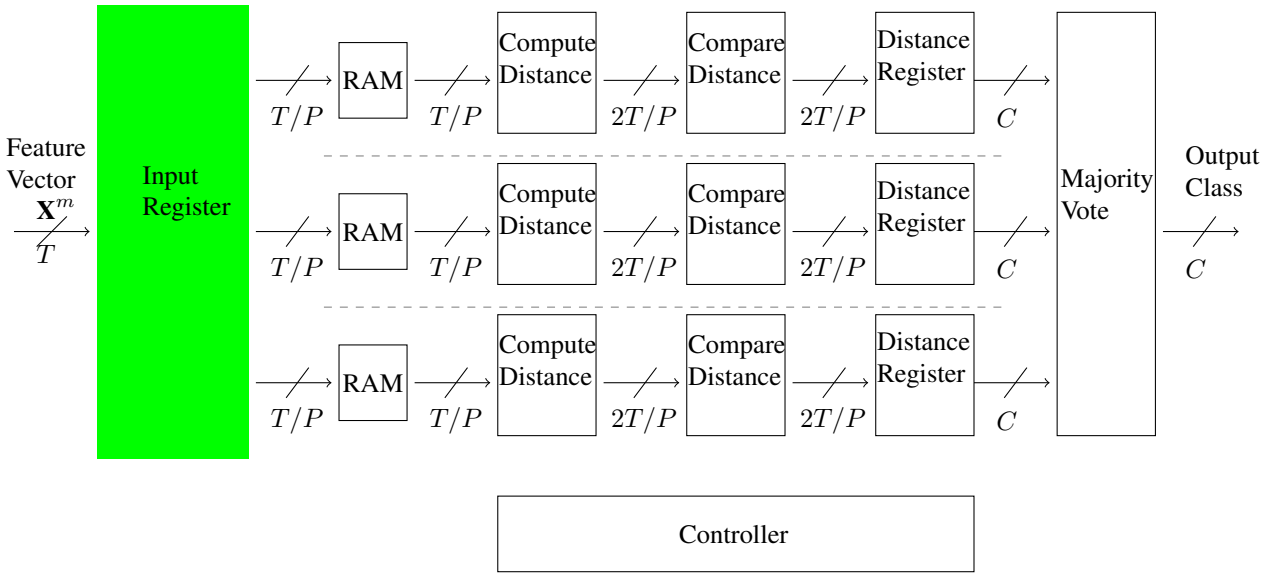

Figure 2. Hardware architecture of ILOC

[4] Yoojin Choi, Mostafa El-Khamy, and Jungwon Lee, "Towards the limit of network quantization," *arXiv preprint arXiv:1612.01543*, 2016.

[5] Jiantao Qiu, Jie Wang, Song Yao, Kaiyuan Guo, Boxun Li, Erjin Zhou, Jincheng Yu, Tianqi Tang, Ningyi Xu, Sen Song, et al., "Going deeper with embedded fpga platform for convolutional neural network," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2016, pp. 26–35.

[6] Naveen Suda, Vikas Chandra, Ganesh Dasika, Abinash Mohanty, Yufei Ma, Sarma Vrudhula, Jae-sun Seo, and Yu Cao, "Throughput-optimized opencl-based fpga accelerator for large-scale convolutional neural networks," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2016, pp. 16–25.

[7] Robi Polikar, Lalita Upda, Satish S Upda, and Vasant Honavar, "Learn++: An incremental learning algorithm for supervised neural networks," *IEEE transactions on systems, man, and cybernetics, part C (applications and reviews)*, vol. 31, no. 4, pp. 497–508, 2001.

[8] Nadeem Ahmed Syed, Syed Huan, Liu Kah, and Kay Sung, "Incremental learning with support vector machines," 1999.

[9] Tomaso Poggio and Gert Cauwenberghs, "Incremental and decremental support vector machine learning," *Advances in neural information processing systems*, vol. 13, pp. 409, 2001.

[10] Yu Sun, Ke Tang, Leandro L Minku, Shuo Wang, and Xin Yao, "Online ensemble learning of data streams with gradually evolved classes," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 6, pp. 1532–1545, 2016.

[11] Ghouthi Boukli Hacene, Vincent Gripon, Nicolas Farrugia, Matthieu Arzel, and Michel Jezequel, "Incremental learning with pre-trained convolutional neural networks and binary associative memories," 2017.

[12] Gian Marco Bo, Daniele D Caviglia, and Maurizio Valle, "An on-chip learning neural network," in *Neural Networks, 2000. IJCNN 2000, Proceedings of the IEEE-INNS-ENNS International Joint Conference on*. IEEE, 2000, vol. 4, pp. 66–71.

[13] Kyunghee Kang and Tadashi Shibata, "An on-chip-trainable gaussian-kernel analog support vector machine," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 57, no. 7, pp. 1513–1524, 2010.

[14] Markos Papadonikolakis and Christos-Savvas Bouganis, "A novel fpga-based svm classifier," in *Field-Programmable Technology (FPT), 2010 International Conference on*. IEEE, 2010, pp. 283–286.

[15] Robi Polikar, Lalita Udpa, Satish S Udpa, and Vasant Honavar, "Learn++: an incremental learning algorithm for multilayer perceptron networks," in *Acoustics, Speech, and Signal Processing. ICASSP'00. Proceedings.IEEE International Conference on*. IEEE, 2000, vol. 6, pp. 3414–3417.

[16] Nikola Kasabov, *Evolving connectionist systems: Methods and applications in bioinformatics, brain study and intelligent machines*, Springer Science & Business Media, 2013.

[17] Robert M French, "Catastrophic forgetting in connectionist networks," *Trends in cognitive sciences*, vol. 3, no. 4, pp. 128–135, 1999.

[18] Zeki Erdem, Robi Polikar, Fikret Gurgen, and Nejat Yumusak, "Ensemble of svms for incremental learning," in *International Workshop on Multiple Classifier Systems*. Springer, 2005, pp. 246–256.

[19] José Fernando García Molina, Lei Zheng, Metin Sertdemir, Dietmar J Dinter, Stefan Schönberg, and Matthias Rädle, "Incremental learning with svm for multimodal classification of prostatic adenocarcinoma," *PloS one*, vol. 9, no. 4, pp. e93600, 2014.

[20] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[21] Herve Jegou, Matthijs Douze, and Cordelia Schmid, "Product quantization for nearest neighbor search," *IEEE transactions on pattern analysis and machine intelligence*, vol. 33, no. 1, pp. 117–128, 2011.

[22] Ian J Goodfellow, Mehdi Mirza, Da Xiao, Aaron Courville, and Yoshua Bengio, "An empirical investigation of catastrophic forgetting in gradient-based neural networks," *arXiv preprint arXiv:1312.6211*, 2013.

[23] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna, "Rethinking the inception architecture for computer vision," *arXiv preprint arXiv:1512.00567*, 2015.