Incremental Learning on Chip

Ghouthi Boukli Hacene, Vincent Gripon, Nicolas Farrugia, Matthieu Arzel and Michel Jezequel
IMT Atlantique, Brest, France
name.surname@imt-atlantique.fr

***Abstract*—Learning on chip (LOC) is a challenging problem in which an embedded system learns a model and uses it to process and classify unknown data, while adapting to new observations or classes. It may require intensive computational power to adapt to new data, leading to a complex hardware implementation. We address this issue by introducing an incremental learning method based on the combination of a pre-trained Convolutional Neural Network (CNN) and majority votes, using Product Quantizing (PQ) as a bridge between them. We detail a hardware implementation of the proposed method (validated on a FPGA target) using limited hardware resources while providing substantial processing acceleration compared to a CPU counterpart.**

***Index Terms*—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 many challenges in the field of machine learning. In particular, Convolutional Neural Networks (CNNs) exhibit outstanding performance in object recognition in images. 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 to factorize their weights [1][2]. Other methods proposed to binarize DNNs weights [3], [4], as well as activation functions [5], with the aim to reduce both DNNs size and computational complexity. These methods allow the implementation of DNNs on embeded systems such as FPGAs [6][7]. However the proposed hardware implementations focus only on the inference process of pre-trained DNNs, assuming that the training procedure has been performed beforehand. They therefore are not compliant with learning on chip (LOC).

LOC allows an embedded system to train a model and to use it to classify new data. It is a challenging research field because of the intensive computation required during the training phases which cannot be handled by a small embedded system with a limited power. Of particular interest are incremental learning methods that aim to learn data sequentially, adapting the model to the new data, and are able to learn new examples and classes while not requiring access to the old data to retrain the model [8]. Although solutions 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 [9], [10], and reuse some prior data for the retraining process [8], [11], which is computationally expensive and does not meet the embedded systems requirements. In [12], the autors introduced a new simple incremental learning method based on transfer learning, product quantization and majority vote (cf. Figure 1). This method adapts the model to new observed examples and classes without retraining or accessing previously processed data, and uses much less computational power than existing counterparts. It is still able to approach 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 incremental learning on chip (ILOC) solution based on [12], with the following claims:

- It is possible to adapt the model to new data (from scratch) without retraining it,
- It uses limited computational resources.

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 by itself, then process and classify new unknown data. Some previous works have proposed solutions to train neural networks [13] on FPGA. However they require to implement gradient descent which is computationally expensive and quickly becomes a bottleneck when the network size increases. Other works propose to train Support Vector Machines on FPGA [14], but still require intensive computational power and large memory usage to store all training data. The need of intensive computation and memory usage during the learning phase represents a major drawback for LOC. In the past few years, a lot of interest has been devoted to big databases such as ImageNet, leading to large DNNs in order to achieve good accuracy. As the implementation of large DNNs is problematic, most existing works focus on the inference process and assume that the learning procedure is performed offline on an external server [6][7], or is using already trained SVMs [15].

Incremental learning allows learning data sequentially and is able to handle new data and new classes without the need to retrain the whole system [16]. Existing solutions propose to add new classifiers to accommodate new data, such as the learn++ method [8], [11] or retrain the model using newly received data together with the old model [9], [10]. To avoid training a large number of classifiers, and to address the *catastrophic forgetting* problem [17], [18], a combination between SVMs and the learn++ method, called "SVMlearn++" [19] was proposed, showing promising improvements [20]. However, this method still needs to retrain a new SVM each time new data are 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 [12], the authors introduced an incremental learning model in which the learning process consists in making a random sampling over input vectors, and then splitting and storing 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.

III. Overview of the Incremental Method

The incremental method introduced in [12] 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 [21], 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 [22]. We chose product random sampling as it achieves good performance, yet it is computationally much lighter than other PQ techniques such as K-means. 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 $\mathbf{y}_p^k$, $\exists \mathbf{x}^m \in X, \mathbf{x}_p^m = \mathbf{y}_p^k$.

Learning data amounts to adding new anchor points to the memory, while remembering the class associated with each of them. Most of the training data is actually disregarded, not adding any anchor point to the memory: a parameter K controls how many anchor points are added for each class. Processing new data amounts to finding the anchor points $\left(\mathbf{q}_p^m\right)_{1\leq p\leq P}$ which are obtained from product quantization of $\mathbf{x}^m$ (cf. Figure 1 step 2), then looking at the corresponding classes $(\tilde{c}_i)_{1\leq i\leq C}$ where C is the number of classes, and performing majority vote to take a decision (cf. Figure 1 step 3). The combination of the pre-trained CNN as feature extractor and the majority vote as classifier allows example and class incremental learning without damaging previously learned knowledge [23] or needing to retrain the model. These properties make the proposed method a good match for embedded system.

The method was tested on challenging vision datasets (CIFAR10 and ImageNet), using Inception V3 [24] as feature extractor. The test gave promising results with 82% of accuracy for CIFAR10 and 92% 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 step 2 and 3, and we assume that step 1 (feature extractor) is performed by an external CPU, providing feature vectors $\mathbf{x}^m$ to the FPGA.

A. Data Quantization

In addition to the quantization of the feature vectors $\mathbf{x}^m$ as described in the method, we quantize the corresponding coordinates using a signed fixed point representation on n bits with 5 bits for the integral part. The main motivation of this step is to adapt data to optimally use the FPGA ressources (e.g. 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 instead of 2).

B. Architecture

The proposed architecture handles both learning and classification processes. The learning process is quite simple, as described in [12]. After random sampling, learning requires splitting the feacture vectors $\mathbf{x}^m$ into P parts, then storing each part $\mathbf{x}_p^m$ into $RAMD_p$, representing anchor vectors denoted $\mathbf{y}_p^m$, and the input class vector $\mathbf{c}^m$ into $RAMC_p$ $(1 \leq p \leq P)$. $RAMD_p$ and $RAMC_p$ are concatenated into one block denoted RAM (cf. Figure 2). 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 2). The processing hardware architecture is made of P identical parts. For each part p, 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 (Compute Distance block). We do the same process with each $\left(\mathbf{y}_p^k\right)_{1\leq k\leq K}$ and compare the obtained result with the distance stored in the register r_p (Compare Distance block). We store the smallest distance and the vector class $\mathbf{c}^p$ one hot encoded on C bits, where C is the number of classes, corresponding to $\mathbf{y}_p^k$ which is the nearest from $\mathbf{x}_p^m$ (Distance Register block). 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 ran 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 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,

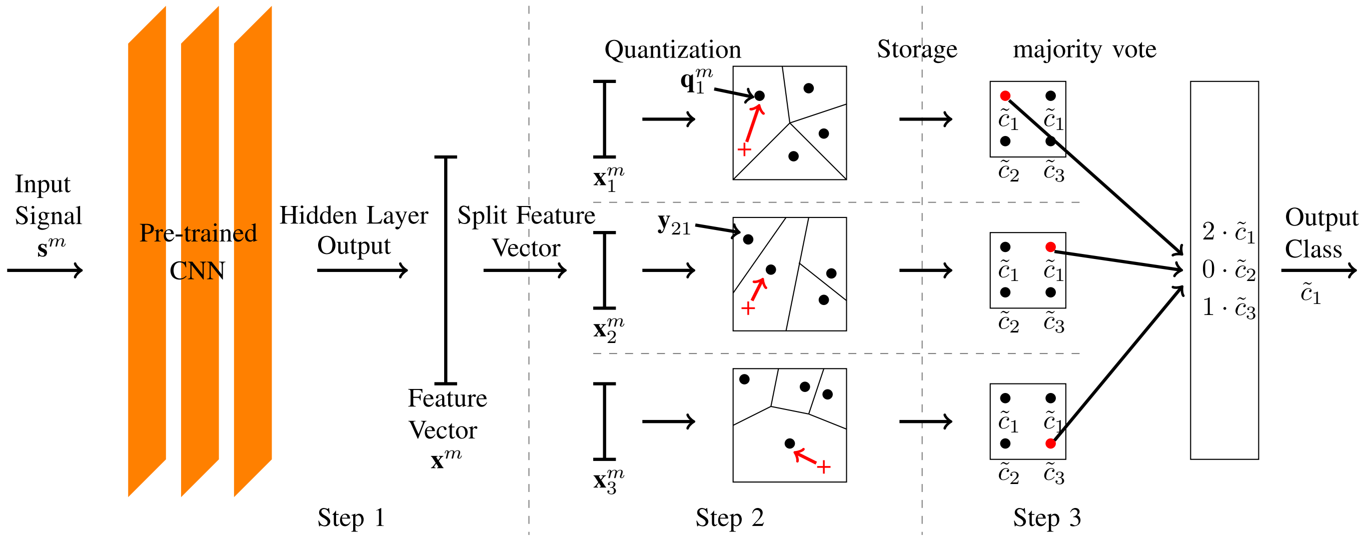

Figure 1. Overview of the proposed method, comprising three main steps. Given a set of samples, we first use a pre-trained CNN for feature extraction (Step 1). Subsequently, we use a PQ technique to quantize the feature vectors (Step 2). Finally, we use a majority vote to classify the quantized data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

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 $\mathbf{x}^m$.

The architecture is fully pipelined, so the number of clock cycles to process an input data is the number needed to 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 CIFAR10 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 [12] to get an accuracy of 82% and 81.6% for 32 and 16 values encoding respectively. To obtain feature vectors we use inception V3 [24] which gives a 2048 dimensional feature vector. 2048 DSPs 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 (estimated by the tool) 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. Table I shows a summary of the resource allocation of the FPGA for the implementation of our proposed architecture.

Table I
FPGA results for the our proposed architecture on Virtex 7 (xc7vx690tffg1157) ($T = 2048$, $P = 64$, $K = 200$ 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. It is able to train a model from scratch. 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 evolve dynamically and to easily adapt to new changes. This architecture allows to perform both training the model and inferring the class of new data. Future work will focus on proposing a hardware implementation for the deep CNN which acts as feature extractor to obtain a complete embedded 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.

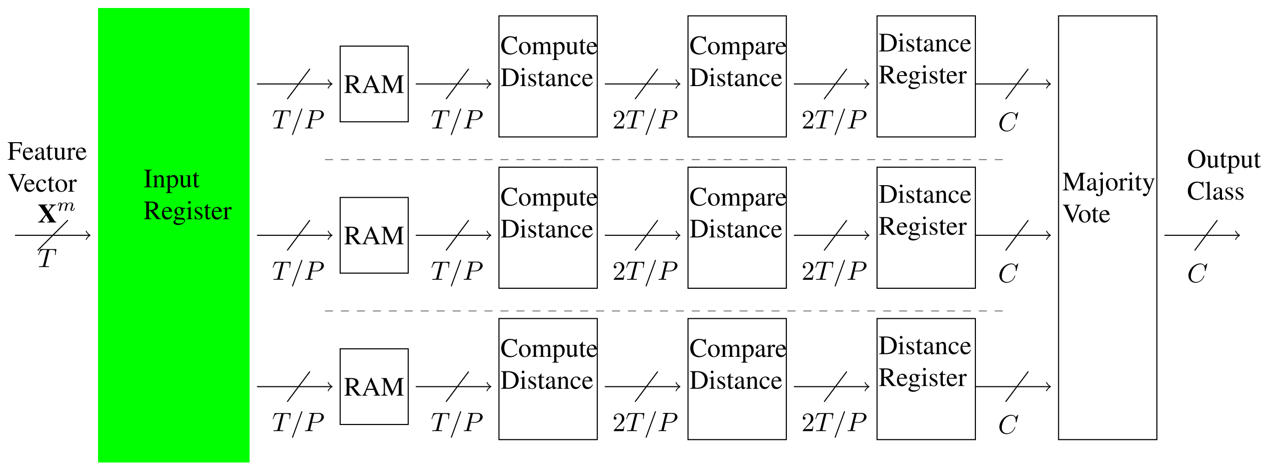

Figure 2. Hardware architecture of ILOC

[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.

[4] Guillaume Soulié, Vincent Gripon, and Maëlys Robert, "Compression of deep neural networks on the fly," in *International Conference on Artificial Neural Networks*. Springer, 2016, pp. 153–160.

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

[6] 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.

[7] 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.

[8] 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.

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

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

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

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

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

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

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