Architectures and Algorithms for User Customization of CNNs

Barend Harris[1], Mansureh S. Moghaddam[1], Duseok Kang[1], Inpyo Bae[1], Euiseok Kim[1]
Hyemi Min[1], Hansu Cho[2], Sukjin Kim[2], Bernhard Egger[1], Soonhoi Ha[1], Kiyoung Choi[1]

[1]Seoul National University [2]Samsung Electronics, Seoul, Korea

Abstract— In this paper we present a convolutional neural network architecture that supports user customization through incremental transfer learning. The architecture consists of a large basic inference engine and a small augmenting engine. After training the basic inference engine and augmenting engine on a large general dataset, the basic inference engine is fixed. For user customization, only the augmenting engine is re-trained on-device using a small user specific dataset provided by the user. To accelerate the training of the augmenting engine we map this to a coarse-grained reconfigurable array processor. The complete network architecture is evaluated using the Caffe framework, and a C-code equivalent network is implemented and tested on a CGRA processor. Experiments with NIST '19 and our user-specific datasets show an increase in accuracy of the system from 76.3% to 93.2% after user customization. Mapping this code to a CGRA gives us a speed up of 45x and a 49- and 3-fold reduced energy consumption over an ARMv7 processor and a 3-way VLIW processor, respectively, showing the potential of CGRAs as DNN processors.

I. Introduction

Over the past few years, we have seen an unprecedented performance improvement of Deep Neural Networks (DNNs), more specifically, Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) in various fields such as computer vision, speech recognition, and others. This leap forward has been made possible by a dramatic improvement in computing power as well as the development of key enabling techniques for artificial neural networks such as convolutional networks and effective ways of learning through backpropagation. Today, DNNs outperform humans not only in terms of speed but also accuracy in various complex tasks [17, 32]. Graphics Processing Units (GPUs) [23] are a representative example responsible for accelerating the computation of DNNs. In particular, the acceleration of the training process with a large dataset, which can take days or even weeks, the use of GPUs and more specialized hardware such as FPGAs or ASICS seems indispensable in this regard.

Adaptation of DNNs in embedded systems has been limited mainly to inference, that is, the task of classifying a specific input through a well-trained DNN. Energy-efficient inference engines are typically provided by customized FPGAs [38] or ASICs [4]. The use of the embedded mobile GPU [10] for inference is possible, however, the relatively high cost in terms of energy consumption limits its practicality. Training DNNs requires (1) a well-defined big dataset of training data and (2) a lot of computing power. Therefore, the typical deployment of DNNs to embedded devices is as follows: the service provider designs and trains a DNN with a large training set in a data center, providing the necessary computational power. Then an energy-efficient inference engine reflecting the trained DNN can be built and deployed with the devices. Alternatively, inference can also be offloaded to the server by sending user input to the provider's data center for processing.

The storage and computational limitations of DNNs, plus the fact that the input dataset is not specific to the user of the embedded device but rather an accumulation of data collected from various individuals, renders on-device learning a challenging task. Standard large models trained on such a general dataset and embedded in devices perform relatively well for their testing set but are not, in fact, tailored to the individual user of the device. We aim to show that by adapting a large general model to a specific user on the device we can achieve a significantly higher overall accuracy for that user.

In this paper, we present a technique that enables on-device personalization of pre-trained large DNNs. In the proposed architecture, we augment an existing large *basic inference engine* (BIE) with a small, task-specific network and a result aggregation layer. The task-specific network and the aggregation layer constitute the so-called *augmenting engine* (AE). Inputs are processed in parallel by both networks, and the aggregation layer combines and generates the final result. The BIE and AE are pre-trained as usual in the service provider's data centers on general data, but once shipped, only the AE is re-trained on the device with user data. Experiments with the publicly available dataset of handwritten characters NIST '19 [13] and our user data show that this technique is able to improve classification accuracy for individual users from 76.3% to 93.2% with a minimal computational overhead of around two percent.

We envisage that future mobile devices will be equipped with either a dedicated NN processing chip or a mobile GPU. We anticipate that the BIE will be executed on one of these or will be implemented in hardware. The overhead of re-training a large network on a small number of images to personalize to a user would be large, or impossible to do if the network is implemented in hardware. Therefore, we propose mapping the re-trainable portion of our network (AE) to a general purpose accelerator commonly used in mobile devices. For this work, we choose the Samsung Reconfigurable Processor (SRP) [33], a commercial coarse-grained reconfigurable array (CGRA) processor, and optimized it for processing DNNs. CGRAs are an attractive alternative to FPGAs or ASICs in high-performance embedded devices such as smartphones thanks to their flexibility and availability. The SRP employed in this work comprises 4x4 processing elements and supports eight simultaneous memory accesses per cycle. The compiler-optimized, software-pipelined DNN loops achieve a high efficiency on this

CGRA chip yielding a 45x speedup and a 49- and 3-fold reduced energy consumption over an ARMv7 processor and a 3-way VLIW processor, respectively.

In summary, the main contributions of our work include:

- We present a novel idea of combining a big basic inference engine (BIE) with a small augmenting engine (AE) and a training–personalization method where the BIE and AE engines are trained offline with a large set of general data but only the AE is re-trained on-device using a small set of user-specific data.
- We show that coarse-grained reconfigurable array (CGRA) processors can be used as efficient accelerators for DNNs and may be a viable alternative to dedicated hardware.
- Through various experiments the proposed method shows a significant improvement in accuracy for user-specific data compared to conventional architectures and/or conventional training algorithms.

The rest of this paper is organized as follows. We give a simple motivational example in Section II. The proposed structure consisting of the basic inference and augmenting engines is defined in Section III. Section IV details the implementation into a production-quality CGRA, the Samsung Reconfigurable Processor (SRP) and Section V describes our experimental setup. Then we evaluate our proposed approach in Section VI. We provide a brief review of related techniques in Section VII and conclude the paper in Section VIII.

II. Motivation

Consider a mobile device such as a smartphone that includes a system to recognize hand-written letters and digits. The system is implemented with a CNN and trained using a large set of training data. Training of the CNN is performed by the vendor, and testing at the factory yields maximum accuracy levels of around 88% [8]. The character-recognition system is then embedded in the device in form of a dedicated accelerator or as an online service, and the device is shipped and sold to the end user - this is the prevalent business model of today's mobile device vendors or service providers that offer DNN-enabled functionality. However, when the end user uses the system with their own handwriting it may result a disappointingly low character recognition accuracy, especially if they have a unique writing style. This is because a general model trained on general data can not take into account all possible variations in user writing style.

We conducted experiments with a well-researched and explored problem: recognition of handwritten letters and digits. Here we use the NIST Special Database 19 (NIST '19) dataset [13] for the necessary training data. NIST '19 is a collection of over 730'000 letters and digits for training and 82'000 for testing. For character classification, we use the LeNet-5 [19] model modified to produce 62 outputs, yielding a test accuracy of 82.1% for the 62 classes (26 upper-, 26 lowercase letters, and 10 digits). For collected individual user data, the average recognition accuracy drops to 76.3% (Figure 1),

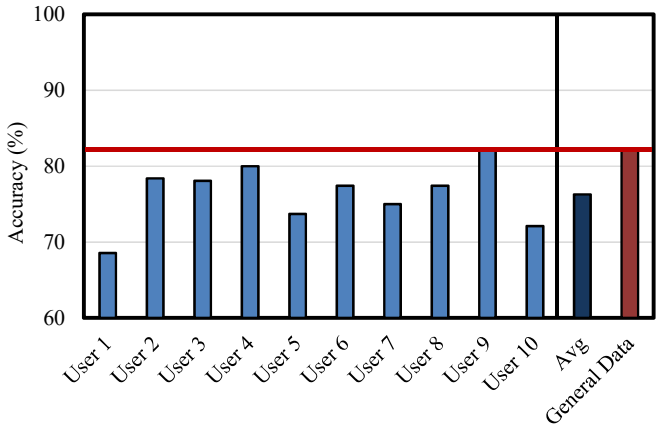

Fig. 1. Accuracy of the general NIST '19 model on user data

far from a satisfactory situation. Figure 1 also shows that for some users the general model achieves a far lower accuracy of around 70% accuracy. This demonstrates that for individual users a general model can not classify their data satisfactorily.

Retraining the large general network model on-device with user data is also difficult, due to the high computational overhead, the limited number of training sets, and the slow adaptation to the user. More sophisticated CNNs trained with more data should be able to achieve higher accuracy on unseen data, however, such systems would not only be harder to train but also more difficult to integrate in embedded systems due to the increased computational and memory requirements of the network. Additionally, there are inherent limitations to the general model's accuracy. For example, in this domain the best performing model has an accuracy of around 88%. This can be a problem in many domains where the training set can not possibly cover every real life scenario.

What is needed, and what we present in this paper, is a system that is based on a powerful general classifier but can be adapted to a specific user with little additional training overhead in terms of both added complexity and the number of required training inputs. Such a system, employed in personal mobile devices, must be able to quickly learn the characteristics of the end user's handwriting on the device itself in order to provide satisfactory recognition accuracy, higher than that of the general classifier.

The next section describes the general architecture of the proposed network that allows retraining of only a part of the network with a limited number of user inputs, enabling on-device personalized classifiers.

III. Methodology

A. The Basic Inference Engine

We first define the targeted usage scenario of this work. A mobile device is to be enhanced by a classifier system that is capable of classifying images according to predetermined categories. In one of its simpler instantiations, such a classifier could be used to recognize handwritten characters. We assume that the vendor of the device has such as system available, either in the form of a dedicated hardware accelerator or as a software implementation. We call this system the *basic inference engine* (BIE). For our purposes, the BIE is a black box

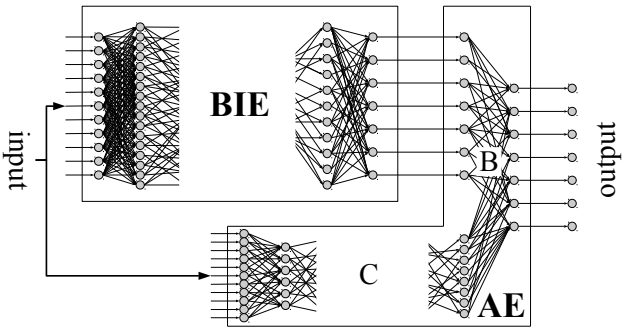

Fig. 2. User customization: augmenting the BIE with the AE.

that takes an image as its input and outputs a probability for each recognized class indicating the likelihood that the input belongs to said category. The vendor has at their availability a big dataset comprising a large number of training data to train the BIE. The BIE is trained once using the dataset and then included in the mobile devices (i.e., by storing its structure plus the weights and biases of the network). For our purposes, the BIE does not have to be retrainable and can be implemented as a hardware accelerator or in software. We anticipate that most likely it will be implemented in software and accelerated using a conventional DNN accelerator such as a GPU or dedicated NN accelerator. The same trained BIE is put into all devices, and the devices are then sold to their end users. Such as system suffers from the problem described in the previous section, namely, that performing well in the general case does not necessarily mean good accuracy for individual users.

B. The Augmenting Engine

To enable user specialization on the device, we augment the BIE with an *augmenting engine* (AE) as shown in Figure 2. The AE itself comprises two parts: a simple convolutional network, labeled `C`, and a fully-connected layer labeled `B`. The basic idea behind this novel system structure is as follows. Firstly, the complete system is well trained on a large set of general data, this covers the recognition of general (i.e., non-user specific) data. The BIE is then fixed and blocks `B` and `C` are retrained on the user specific data. We train `B` and `C` initially as well as the BIE instead of initiating the weights randomly, as this ensures they are initialized to values appropriate for a similar problem (i.e. the general task), so less time will be needed to adapt to the user data. The purpose of the `C` block is to learn the user data, and the function of the`B` block is to learn to combine the outputs of the BIE and the `C` block adaptively.

C. Initial Training and Personalization

The augmented BIE needs to receive initial training in order to produce sensible outputs from the start. We firstly train the BIE and AE together, the BIE is then fixed (we use the term "fix" to refer to setting the learning rate of a network to zero, thereby freezing its weights and bias values). The left-hand side of Figure 3 illustrates the initial training at the manufacturing side.

After the initial training, the system is either embedded in hardware or is implemented in software and accelerated with a GPU or neural processor on the mobile device. The BIE is

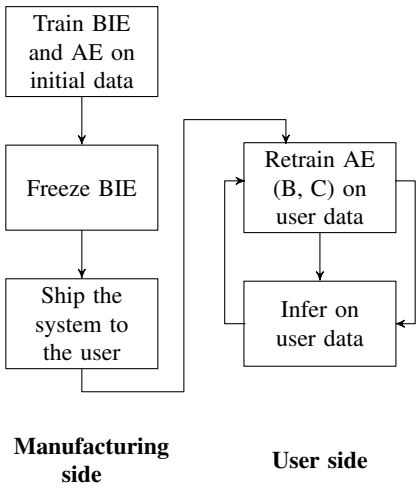

Fig. 3. Training flow.

fixed, but supervised learning is enabled for the AE, i.e., blocks `B` and `C`. Through using the classifier to recognize handwritten characters, the user continuously trains the system. For an application that converts handwritten notes into text, we can use a language model, such as the one used in current predictive text on smartphones, to gather labels for misclassified and correctly classified characters. When the user selects a word in the suggested list, we can use this information to label our data and thus perform supervised learning. Characters that are not corrected are assumed to be classified correctly, while corrections represent the misclassified characters. A decaying learning rate can be applied to the entire AE block, to ensure quick adaptation at the beginning. We use an exponentially decaying learning rate of 0.01 for the entire AE block. This user-specialization process is shown on the right-hand side of Figure 3.

IV. Processing Neural Nets with CGRAs

To accelerate the training of the AE, we propose the use of coarse-grained reconfigurable array (CGRA) processors. These have been proposed as an alternative to FPGAs and ASICs in the embedded system domain. CGRAs are are a bit less efficient in terms of power-consumption or computational power when executing a specific task compared to the other contenders, however, the benefit of being programmable to perform a variety of distinct tasks allows hardware manufacturers to replace several task-specific ASICs with one programmable CGRA. Samsung Electronics, for example, has been integrating a commercial versions of their Samsung Reconfigurable Processor (SRP) [33] into their MPSoCs for smartphones, TVs, printers, and cameras for several years now [29, 20, 21, 31].

Given that CGRAs excel at data-flow-style processing of loops, it is not surprising that the basic architectural structure of a CGRA is similar to many of the recently proposed hardware accelerators for efficient DNN processing [6, 14, 27]. DNN applications consist mainly of matrix multiplications, these in turn consist of many high iteration loops with simple loop bodies, which are ideal candidates for software pipelining.

Here we describe architectural and compiler changes to map DNNs efficiently to CGRAs; results of this mapping are presented in Section VI.

A. Adapting CGRAs to Execution of DNNs

The CGRA processor comprises an array of processing elements (PEs) connected by an irregular interconnection network. PEs are typically heterogeneous in their functionality. Temporary data storage is provided by register files (RFs), and fast on-chip data storage is provided by SRAM. The CGRA used in this work operates in data-flow mode. Loops are converted into a software-pipelined modulo-scheduled representation by a CGRA compiler [24].

The Samsung Reconfigurable processor employed for this work is a hybrid VLIW/CGRA processor based on the ADRES architecture [22]. The SRP allows fast switching between CGRA and VLIW mode in the order of 1-3 clock cycles. Software-pipelinable loops are executed on all PEs of the CGRA while for control-flow intensive code the processor executes in VLIW mode. A C compiler automatically generates code for both modes and inserts the necessary glue code between the two. Figure 4 shows a schematic diagram of the SRP CGRA processor.

CGRAs show excellent performance when executing regular and computationally intensive loops, yet to execute DNNs efficiently, the domain-specific challenge of getting the data of the biases and weight tables into the PEs at a sufficient bandwidth require minor modifications to the architecture. The base configuration of the SRP is a 32-bit floating-point CGRA with 4x4 heterogeneous PEs and a total of 320KB of on-chip data SRAM. In this original configuration, four PEs are connected to the data memory, and half of the 16 PEs support floating point operations. To accommodate for the higher bandwidth requirements of DNN processing, four more PEs are designated to support memory operations, yielding a maximal theoretical throughput of 8 load/store operations per clock cycle. The on-chip data store is implemented by eight banks of SRAM. To avoid stalls caused by bank conflicts during execution of CGRA code, the latency of the load instruction is set to 16 cycles from the original six. Since the actual memory access latency is four cycles, the remaining 12 cycles allow the so-called *data memory queue* (DMQ) to buffer and serialize memory accesses that are directed to the same memory bank. This increased memory latency has only a minimal effect on the overall execution time because load latency can be almost completely hidden in software-pipelined loops.

No other modifications were made to the SRP. Support for fused multiply-add operations has not proven to be effective. This again is because loops are software-pipelined; the performance limiting factor is not how long a single loop iteration takes but how frequently a new loop iteration can be started, i.e., the so-called, the initiation interval (II).

B. A DNN Framework for CGRAs

Code executed on the CGRA runs bare-metal and as a consequence support for runtime environments (i.e., Python) or libraries (such as the C standard library or math libraries) is not available. Existing frameworks heavily depend on libraries (Caffe [16], Tensorflow [35], Darknet [28]) and some do not support the parallel network structure proposed in this paper.

We have chosen to implement a simple deep neural network framework written in pure C. The framework does not have any external dependencies in order to support embedded accelerators such as CGRAs. The implementation of certain activation functions, especially the gradient computations, have been taken from Darknet [28]. The framework supports Caffe-style input files, including networks with multiple branches. We have currently implemented the following layer types: Input, Convolutional, Fully Connected, Max Pooling and Reshape layers.

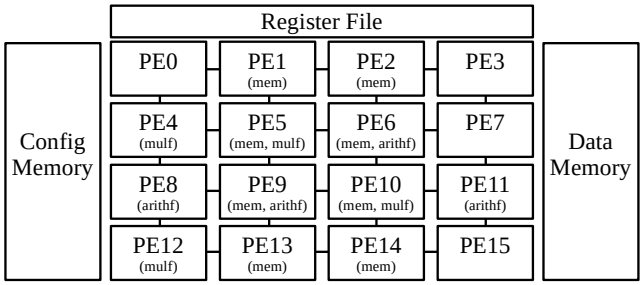

Fig. 4. Schematic of a coarse-grained reconfigurable array processor.

The functionality of these layers is identical to those in other frameworks. The *reshape layer* is used to concatenate the output of two different layers or to change the shape of a layer.

The framework's workflow is as follows: Firstly, the input dataset, input labels, and the network specification are read as inputs. Then, the framework parses the network specification to construct the desired network as a C program. The framework executes this program, performing forward propagation, back propagation and weight updates multiple times on the network, based on the supplied inputs and labels. Finally, the framework outputs the accuracy and loss of the trained model and saves the weight information to an external file.

C. Code Optimizations

Only innermost loops without control flow (`if-else` or function calls) can be modulo-scheduled and mapped to the CGRA. We maximize the code that can be mapped to the CGRA by performing a number of compiler-level optimizations on the code generated by the framework. Several loop transformation techniques are employed to increase the scope of the innermost loop, specifically loop unrolling, loop fusion, and loop interchange. For loop unrolling, we estimate the ideal unrolling factor for each loop at every loop nest level in order to maximize the parallel usage of the PEs and unroll accordingly. For loop fusion, if there are several loops where the same memory locations are accessed in identical order, the bodies of these loops are fused in order to reduce the overhead of accessing memory multiple times. Loop interchange is applied after performing the other loop transformations. We reorder loops that have no loop-carried dependencies in a way such that the loop with the highest number of iterations becomes the innermost loop, thereby maximizing the code mapped to the CGRA. These transformations are performed along with the existing if-conversion and inlining optimizations [18].

V. Experimental Setup

A. Datasets

We evaluate the proposed design on the classification problem of recognizing handwritten letters and digits. Training is performed as described in Section III-C using the NIST '19 [13] training database. User-specific training is performed using handwritten character data gathered by the authors of this paper.

General Dataset: The NIST '19 dataset consists of a total of 62 classes: lower-case characters "a"-"z", upper-case characters "A"-"Z", and the digits "0"-"9". These can further be divided into five categories `all`, `letters`, `lower`, `upper` and `digits`, each containing the expected subset of the data. NIST '19 consists of 731,668 and 82,587 images for training and testing respectively.

User Specific Dataset: User data for these 62 classes was also collected from 10 users using an Android app developed by the authors of this paper. We collected 40 images of each of the 62 alpanumeric characters from every user. We use 10 as the test set and the remaining 30 as the training set. This results in a total of 1860 training and 620 testing images per user.

Both NIST '19 and user-specific images are pre-processed using similar techniques used in preprocessing the EMNIST dataset [9]. The characters have a gaussian smoothing filter applied, are centered, padded, and scaled down to a size of 28x28 pixels. We do not directly use the EMNIST dataset for training as we wished to ensure that the exact same preprocessing procedure was applied to both the general and the user-specific datasets.

B. BIE and AE Network Structure

For the BIE we use a version of LeNet-5 [19] adapted to produce 62 outputs. In the AE for block `C` we choose a simple convolutional neural network comprised of two pooling layers, one convolutional layer, and a reshape (flattening) layer. The first pooling layer downsamples the 28x28 input to 14x14 pixels, followed by a 10-channel 5x5 convolution. The second pooling layer downsamples the 10 channels to 5x5 resolution each that are then flattened in to a 250-element vector. Block `B` combines the 62-element vector of the BIE with the vector from `C`. After applying a ReLU activation function a fully-connected layer connects the 312 inputs to the 62 output channels.

The overhead of inferencing and training the augmenting engine relative to the BIE is shown in Table I. We observe that both the training and inference overhead are minimal, even in comparison with the relatively simple LeNet-5. Table II shows the overhead of training with one image, assuming a 4-byte size for the weights and activations.

TABLE I
Relative overhead of the AE wrt the BIE (LeNet-5)

	MACC	#Neurons	#Weights
Inference	2%	12%	4%
Training	2%	10%	4%

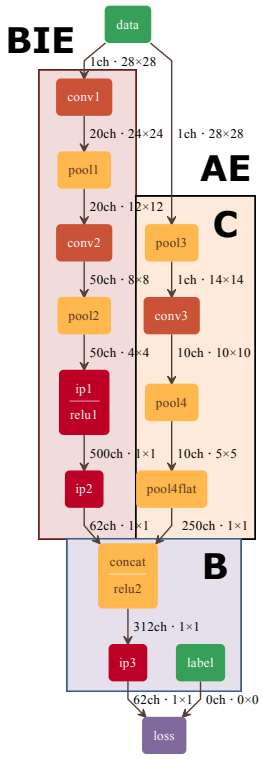

Fig. 5. The BIE (left channel) and the AE (blocks B and C) for NIST.

TABLE II
Overhead of the AE and BIE (LeNet-5) on one image

	MACC	Neurons	Weights
BIE Inference	2,319 k	79 kB	1,826 kB
AE Inference	44 k	9 kB	78 kB
BIE Training	4,167 k	155 kB	3,652 kB
AE Training	83 k	15 kB	157 kB

C. Mapping to CGRA

The SRP [33] employed in this work has been modified as described in Section IV-A. The 16 processing elements (PEs) are arranged on a 4x4 CGRA grid with eight processing elements able to perform memory operations. Four PEs support floating point arithmetic, and four PEs are allocated for floating point multiplication. Figure 4 shows the location and functionality of the different PEs. The VLIW component of the processor uses PEs 0–2 to execute code that cannot be mapped to data-flow mode.

We run the optimized C code generated by our framework from Section IV-C on the modified SRP architecture using a cycle-accurate simulator. We give the result for training the network on one image. We compare this against a cycle-accurate simulation of the 3-issue VLIW processor, simulated in the same manner as the hybrid chip. For both of these we assume a clock speed of 500MHz. Additionally, we compare our result against an ARMv7 processor in a Rasperry Pi 3 with a clock speed of 1.2GHz. The results of the ARMv7 architecture were obtained by executing the generated DNN code directly on the ARM processor. The ARMv7 was chosen as a comparison point as, like our hybrid VLIW/CGRA processor, it is also a widely-used chip in general processing on mobile devices.

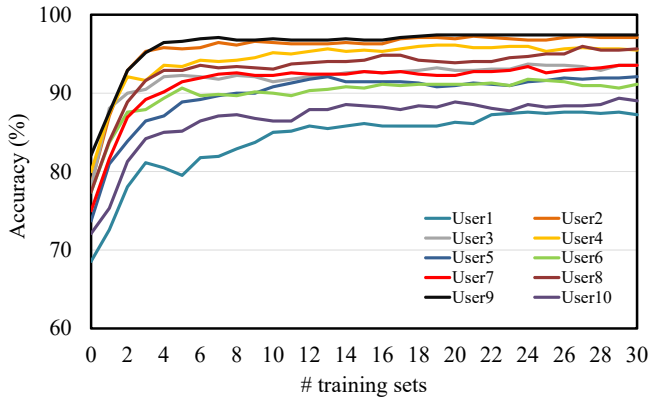

Fig. 6. Accuracy of BIE + AE on user data

VI. Evaluation

In this section, we evaluate the proposed architecture and show the results of executing the system on the modified Samsung Reconfigurable CGRA processor.

A. Architecture Results and Analysis

To see the effect of user customization, the BIE and AE are first trained on the original NIST '19 training set. The weights of the BIE are then frozen to generate the system as it is shipped to the end user. The AE is retrained with an increasing dataset size for a specific user, and the system's accuracy is tested against the user's test set after each training step. Each training set consists of one image from each class, and we train for ten epochs. The accuracy for all users increases with the dataset size as shown in Figure 6. We observe that after just a small number of training sets we achieve a large increase in accuracy, after 5 training sets the average accuracy increases from 76.3% to 90.7%.

Table III lists the results before and after training with the 30 sets of user images for each of the five categories of NIST. We can see that for all the different categories, after retraining the AE on the user data we get a significant increase in accuracy, often surpassing the accuracy of the general model significantly. This shows that the AE can successfully adapt to a user's writing style and provide high levels of accuracy for a particular user, compared the general model trained only on general data. Much of the errors in such a system come from confusing similar glyphs, for example, characters where the lower and upper case characters look similar and are difficult to distinguish (such as f, F) or the characters (i,l,1,j). However, for a single user, there is usually some consistency in the way they write these characters. This accounts for the large improvements in the 'all' and 'letters' categories where many of the misclassified characters are of this type, demonstrating the effectiveness of user customization.

A valid question is if the BIE is needed at all. To test this, the AE, without the BIE, is first trained on the general NIST data, then retrained on the user data. This gives slightly worse but still satisfactory final results for individual users. However, the initial accuracy of this AE-only general model is lower, with a base accuracy of 70% compared to the 76% of the BIE + AE model. Additionally, it takes more training samples until the accuracy reaches a satisfactory level, which is particularly relevant for our scenario, as for end users it is important to have as high accuracy initially and improve this as quickly as possible. Eliminating the BIE also leads to a significantly reduced accuracy on the general data after user customization, falling to 63% for the AE-only model but just to 74% on the BIE + AE model. If we increase the size and complexity of the BIE, we expect these discrepencies to also increase.

B. Mapping to CGRA

Last but not least, we evaluate the mapping of the proposed AE to the SRP optimized for DNNs using our custom compiler. Figure 7 shows that compared to the 3-issue VLIW and an ARMv7 chip as described in Section V-C, when training the AE, the CGRA achieves a speedup of 45x. In terms of energy consumption, the SRP achieves a 49- and 3-fold reduction in energy consumption compared to the ARMv7 processor and the VLIW processor, respectively. We additionally perform the same experiment with the BIE (Adapted LeNet), to demonstrate that the optimizations are not specific just to the AE network structure but can also be applied more generally to DNN acceleration. Here, we also see a similar speedup of 45x compared to the ARM and VLIW processors. Table IV shows the power/energy consumed by the three processors.

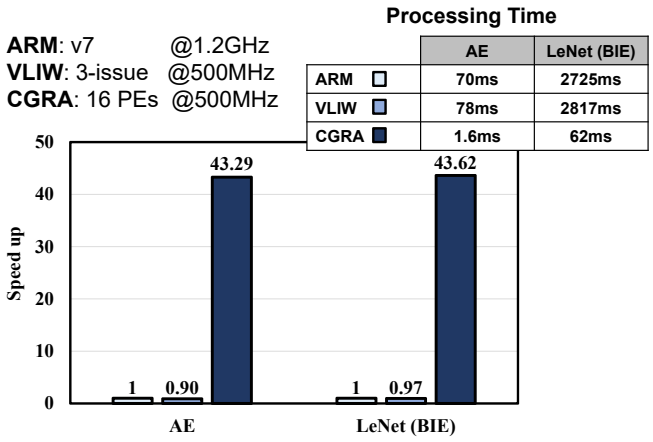

Fig. 7. Mapping DNNs to a coarse-grained reconfigurable array.

VII. Related Work

Our method is an incremental transfer learning based approach. The problem of transfer learning [2, 5, 25], is where the knowledge gained on an initial task is used to improve the performance on a new related task. In our case, the initial task is general data classification, and our new task is user-specific data classification. Two common methods of performing this are Joint Training [3] and Fine-Tuning [12]. In joint training, the entire CNN is retrained with both general and user datasets to optimize it for both tasks, whereas fine-tuning-based techniques retrain a task-specific part of the network. This requires only the new dataset, but often the entire architecture is retrained with a reduced learning rate. As it is impractical to store and train on the original dataset or to retrain the entire architecture on-device, both methods are not easily accommodated to the embedded system domain. Other examples of transfer learning include the Efficient Life-long Learning Algorithm (ELLA) [11] and Deep Transfer Networks (DTN) [39].

TABLE III
CLASSIFICATION ACCURACY BEFORE AND AFTER RETRAINING THE SYSTEM USING USER-SPECIFIC DATASETS.

	all		**letter**		**lower**		**upper**		**digit**	
Dataset	**before**	**after**	**before**	**after**	**before**	**after**	**before**	**after**	**before**	**after**
NIST	82.1	73.9	69.7	73.8	88.8	87.0	96.2	96.1	98.2	96.7
User 1	68.6	87.3	74.6	91.2	86.5	96.2	95.8	98.5	97.0	99.0
User 2	78.4	97.1	78.1	98.7	94.2	100	100	100	91.0	100
User 3	78.1	93.6	76.0	94.8	97.3	98.9	99.2	99.6	98.0	100
User 4	80.0	95.5	78.5	96.5	95.4	98.9	99.2	100	99.0	100
User 5	73.7	92.1	73.3	92.3	85.4	98.9	94.2	98.5	98.0	100
User 6	77.4	91.1	79.2	93.3	97.3	99.2	97.7	100	99.0	100
User 7	75.0	93.5	75.6	96.5	90.8	99.6	100	100	100	100
User 8	77.4	95.6	78.7	96.5	93.1	99.6	98.5	100	100	100
User 9	82.1	97.4	81.2	97.9	95.0	99.2	99.6	100	100	100
User 10	72.1	89.0	72.7	90.4	90.4	98.8	89.6	97.7	93.0	99.0
Average	76.3	**93.2**	76.8	**94.8**	92.5	**98.9**	97.4	**99.4**	97.5	**99.8**

TABLE IV
POWER ESTIMATION FOR TRAINING ON ONE IMAGE

	Average Power [mW]	**Energy** [mJ]
ARMv7 [1]	169	11.83
VLIW	50	3.90
CGRA	150	0.24

These are similar to our approach, but relate to adding new classes and result in large network structures where large parts of the network are retrained on the new task. Whereas we focus on improving the performance of a predetermined number of classes and, as we target the embedded domain, just retrain a small portion of our network (the AE).

Incremental learning [30] is where input data is used to continuously train a DNN model as more data becomes available. The aim of this is learn from new data without losing previous knowledge. One such approach is error driven incremental learning [36]. Here, classes are gradually added to a CNN as they become available. The CNN model is expanded using cloning and branching, resulting in a large number of CNN models linked in a tree like structure. Again this relates to adding new classes rather than increasing accuracy on predetermined classes, and results in a large model unsuitable for the embedded domain.

There has also been some work on personalizing DNN models on a user's data [37]. This was done for generating perzonalized language models using an RNN-LSTM to perform personalized sentence completion in the style of a user and is targeted to mobile devices. This is similar to our work, but focuses on a different application area and network structure. It also makes use of techniques based on fine-tuning rather than the proposed architecture with a large BIE and small AE.

There have been numerous works regarding the acceleration of DNNs on embedded mobile devices. These often make use of FPGAs, but concentrate almost exclusively on the inference task. There have been FPGAs developed specifically for accelerating certain networks, such as SqueezeNet [15] and a network for Korean character recognition [26]. These are often network-specific implementations, and a change in network design would require a different configuration of the FPGA. The difference with our approach is that CGRA acceleration is not network dependant and requires no reconfiguration based on a change in application. Additionally these approaches can be integrated with ours by using the FPGA accelerated network as the BIE. Another architecture for accelerating DNNs on mobile devices is Eyeriss [7] an energy efficient mobile hardware accelerator for NN implementations. This tries to minimize data movement on a spatial architecture similar to a CGRA. However, it is an accelerator specific for CNN applications, whereas what we propose is an adaptation of a general purpose accelerator to also accelerate CNNs. There has even previously been work on accelerating CNNs with CGRAs [34] however, this proposes significant hardware changes to the CGRA, rendering it suitable only for CNN acceleration. Whereas what we propose are minor modifications to the CGRA, allowing us to use it as both an effective CNN and general purpose accelerator.

VIII. CONCLUSIONS

In this paper, we have shown that on device user customization can be important in practical use cases, we introduce a novel network structure to enable this. The entire structure, comprising a large basic inference engine (BIE) and a small augmenting engine (AE), are first trained on a large general dataset by the vendor, user customization is then achieved by retraining just the small AE on the device. The results on handwritten alphanumeric characters show the potential of the proposed system. The large model trained with general data shows an average accuracy of 76.3% for user characters. After user specialization, the proposed structure achieves an increase in average accuracy of 17% to 93.2% at a minimal computational overhead of around 2%. By mapping the AE training to a DNN-optimized CGRA we show the potential of these architectures for DNN processing. Through aggressive loop transformations, the CGRA-enabled code runs up to 45x faster than both a 3-issue VLIW and an ARMv7 processor with a 49- and 3-fold reduced energy consumption, respectively.

REFERENCES

[1] Arm Limited. Arm processor data sheets. https://developer.arm.com/products/processors/, 2017.

[2] A. Arnold, R. Nallapati, and W. W. Cohen. A comparative study of methods for transductive transfer learning. In *Seventh IEEE International Conference on Data Mining Workshops (ICDMW 2007)*, 2007.

[3] R. Caruana. Multitask learning. *Machine Learning*, 28(1):41–75, 1997.

[4] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam. DianNao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. *SIGARCH Comput. Archit. News*, 42(1), 2014.

[5] T. Chen, I. J. Goodfellow, and J. Shlens. Net2Net: Accelerating learning via knowledge transfer. *CoRR*, abs/1511.05641, 2015.

[6] Y.-H. Chen, J. Emer, and V. Sze. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. In *Proceedings of the 43rd International Symposium on Computer Architecture*, ISCA '16, pages 367–379. IEEE Press, 2016.

[7] Y.-H. Chen, J. Emer, and V. Sze. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. *SIGARCH Comput. Archit. News*, 44(3):367–379, June 2016.

[8] D. C. Ciresan, U. Meier, L. M. Gambardella, and J. Schmidhuber. Convolutional neural network committees for handwritten character classification. In *2011 International Conference on Document Analysis and Recognition*, pages 1135–1139, Sept 2011.

[9] G. Cohen, S. Afshar, J. Tapson, and A. van Schaik. EMNIST: an extension of MNIST to handwritten letters. *CoRR*, abs/1702.05373, 2017.

[10] A. Dundar, J. Jin, B. Martini, and E. Culurciello. Embedded streaming deep neural networks accelerator with applications. *IEEE Transactions on Neural Networks and Learning Systems*, 2016.

[11] E. Eaton and P. L. Ruvolo. ELLA: An efficient lifelong learning algorithm. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, volume 28, pages 507–515, 2013.

[12] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2013.

[13] P. J. Grother. NIST special database 19 handprinted forms and characters database. *National Institute of Standards and Technology*, 2016.

[14] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally. EIE: Efficient inference engine on compressed deep neural network. In *Proceedings of the 43rd International Symposium on Computer Architecture*, ISCA '16, pages 243–254, 2016.

[15] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer. SqueezeNet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size. *CoRR*, abs/1602.07360, 2016.

[16] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[17] J. M. Kanter and K. Veeramachaneni. Deep feature synthesis: Towards automating data science endeavors. In *2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, 2015.

[18] K. Kennedy and J. R. Allen. *Optimizing Compilers for Modern Architectures: A Dependence-based Approach*. Morgan Kaufmann, 2002.

[19] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[20] W.-J. Lee, Y. Shin, J. Lee, J.-W. Kim, J.-H. Nah, S. Jung, S. Lee, H.-S. Park, and T.-D. Han. SGRT: a mobile gpu architecture for real-time ray tracing. In *Proceedings of the 5th high-performance graphics conference*, pages 109–119. ACM, 2013.

[21] W.-J. Lee, Y. Shin, J. Lee, J.-W. Kim, J.-H. Nah, H.-S. Park, S. Jung, and S. Lee. A novel mobile GPU architecture based on ray tracing. In *2013 IEEE International Conference on Consumer Electronics (ICCE)*, 2013.

[22] B. Mei, B. Sutter, T. Aa, M. Wouters, A. Kanstein, and S. Dupont. Implementation of a coarse-grained reconfigurable media processor for AVC decoder. *Journal of Signal Processing Systems*, 51(3):225–243, 2008.

[23] NVIDIA. Graphics Processing Unit (GPU). http://www.nvidia.com/object/gpu.html, 2016.

[24] T. Oh, B. Egger, H. Park, and S. Mahlke. Recurrence cycle aware modulo scheduling for coarse-grained reconfigurable architectures. *SIGPLAN Notes*, 44(7):21–30, June 2009.

[25] S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, Oct 2010.

[26] H. Park, C. Lee, H. Lee, Y. Yoo, Y. Park, I. Kim, and K. Yi. Optimizing DCNN FPGA accelerator design for handwritten hangul character recognition: Work-in-progress. In *Proceedings of the 2017 International Conference on Compilers, Architectures and Synthesis for Embedded Systems Companion*, CASES '17, 2017.

[27] R. Prabhakar, Y. Zhang, D. Koeplinger, M. Feldman, T. Zhao, S. Hadjis, A. Pedram, C. Kozyrakis, and K. Olukotun. Plasticine: A reconfigurable architecture for parallel paterns. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ISCA '17, 2017.

[28] J. Redmon. Darknet: Open source neural networks in c. http://pjreddie.com/darknet/, 2013–2016.

[29] http://wwW.samsung.com/us/business/oem-solutions/pdfs/Exynos_v11.pdf, 2011.

[30] Y.-Y. Shen and C.-L. Liu. *Incremental Learning Vector Quantization for Character Recognition with Local Style Consistency*, pages 228–239. Springer International Publishing, 2016.

[31] Y. Shin, J. Lee, W.-J. Lee, S. Ryu, and J. Kim. Full-stream architecture for ray tracing with efficient data transmission. In *2014 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2014.

[32] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[33] D. Suh, K. Kwon, S. Kim, S. Ryu, and J. Kim. Design space exploration and implementation of a high performance and low area coarse grained reconfigurable processor. In *International Conference on Field-Programmable Technology (FPT)*, pages 67–70, 2012.

[34] M. Tanomoto, S. Takamaeda-Yamazaki, J. Yao, and Y. Nakashima. A cgra-based approach for accelerating convolutional neural networks. In *Embedded Multicore/Many-core Systems-on-Chip (MCSoC), 2015 IEEE 9th International Symposium on*, pages 73–80, Sept 2015.

[35] TensorFlow v1.0. https://www.tensorflow.org/, 2017.

[36] T. Xiao, J. Zhang, K. Yang, Y. Peng, and Z. Zhang. Error-driven incremental learning in deep convolutional neural network for large-scale image classification. In *ACM Multimedia*, 2014.

[37] S. Yoon, H. Yun, Y. Kim, G. Park, and K. Jung. Efficient transfer learning schemes for personalized language modeling using recurrent neural network. *CoRR*, abs/1701.03578, 2017.

[38] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong. Optimizing FPGA-based accelerator design for deep convolutional neural networks. In *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '15, pages 161–170. ACM, 2015.

[39] X. Zhang, F. X. Yu, S. Chang, and S. Wang. Deep transfer network: Unsupervised domain adaptation. *CoRR*, abs/1503.00591, 2015.