https://doi.org/10.1016/j.vlsi.2018.11.001

Integration, the VLSI Journal

www.elsevier.com/locate/vlsi

Architectures and algorithms for on-device user customization of CNNs☆

Barend Harris, Inpyo Bae, Bernhard Egger *

Dept. of Computer Science and Engineering, Seoul National University, Seoul, South Korea

ARTICLE INFO

Keywords:
Convolutional neural networks
User customization
On-device learning
Coarse-grained reconfigurable array

ABSTRACT

A convolutional neural network (CNN) architecture supporting on-device user customization is proposed. The network architecture consists of a large CNN trained on a general data and a smaller augmenting network that can be re-trained on-device using a small user-specific data provided by the user. The proposed approach is applied to handwritten character recognition of the Latin and the Korean alphabet, Hangul. Experiments show a 3.5-fold reduction of the prediction error after user customization for both the Latin and the Korean character set compared to the CNN trained with general data. To minimize the energy required when retraining on-device, the use of a coarse-grained reconfigurable array processor (CGRA) in a low-power, efficient manner is presented. The CGRA achieves a speedup of 36× and a 54-fold reduced energy consumption compared to an ARMv8 processor. Compared to a 3-way VLIW processor, a speedup of 42× and a 12-fold energy reduction is observed, demonstrating the potential of general-purpose CGRAs as light-weight DNN accelerators.

1. Introduction

Deep Neural Networks (DNNs), more specifically Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) have been responsible for a large performance increase in a number of complex tasks. Typical applications are computer vision [2,3], handwriting recognition [4,5], and speech recognition [6,7], though ever more applications are being proposed [8,9]. More and more, these application domains move into the embedded system domain, leading to research on adapting DNNs to resource-constrained environments and DNN accelerators [10–12].

When DNNs are used as part of embedded systems, DNN processing is often offloaded to a server [13]. However, with issues such as privacy becoming more of a concern, sending sensitive or personal data to a server for training or inference becomes less appealing and on-device implementations of DNNs gain in importance. Offloading to a server is also not ideal for real-time applications, or where access to a network is not readily available. Furthermore, standard large models trained on 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. This work aims to show that by adapting a large general model to a specific user on the device, a significantly higher overall accuracy for that particular user can be achieved.

There has been significant research into executing DNNs on device, however, this has typically been limited to inference, i.e. classifying an input through a well-trained DNN. Energy-efficient engines are typically provided by customized FPGAs [14] or dedicated ASICs [10], but these are limited to inference. Mobile GPUs are another feasible method of acceleration, however, they have relatively high costs in terms of energy consumption. Coarse-grained reconfigurable arrays (CGRAs) are an existing class of embedded accelerator that share several characteristics with proposed dedicated DNN hardware accelerators. Both consist of a spatial array of Processing Elements (PEs) connected by an interconnection network, and with memory situated close to these PEs. CGRAs have been integrated into embedded MPSoCs to process multimedia workloads for some time now. The Samsung Reconfigurable Processor (SRP) [15], a commercial CGRA, has been integrated into several classes of devices such as cameras, smartphones, tablets, or smart TVs [16–19]. CGRAs provide ample processing power at low power consumption. Even though CGRAs are typically less efficient when executing a specific task compared to ASICs, they have the benefit of being programmable to perform various distinct tasks, allowing hardware

☆ Extension of Conference Paper: this article is an extension of a paper presented at ASP-DAC '18 [1]. Additional material includes an application of the BIE-AE technique to Korean character recognition and more details on the AE design and selection, the CGRA hardware, the compiler optimizations, and the evaluation of the presented method.

* Corresponding author.
E-mail addresses: barend@csap.snu.ac.kr (B. Harris), inpyo@csap.snu.ac.kr (I. Bae), bernhard@csap.snu.ac.kr (B. Egger).

manufacturers to replace several task-specific ASICs with a single programmable CGRA. Presented in this work is an architecture that can exploit the power of CGRAs for retraining, while being able to incorporate pre-trained hardware implementations of larger CNNs or higher power accelerators for inference.

In the proposed network architecture, an existing large *basic inference engine* (BIE) is augmented with a small, task-specific network and a results aggregation layer. The task-specific network and the aggregation layer constitute the *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 a data center on general data, but once shipped, only the AE is re-trained on the device with user data.

Re-training a large network with a small number of user-specific training samples to personalize the network to a user still poses a challenge on resource-constrained embedded devices. Using the smaller augmenting engine allows an existing larger network to be customized to a user without losing generality and with minimal overhead. Experiments with the publicly available dataset of handwritten characters, NIST '19 [20], and our user data show that this technique is able to reduce the classification error for individual users by a factor of 3.5, improving the accuracy from 76.1% to 93.7% with a minimal computational overhead of around two percent compared to retraining the whole network. For the more complex problem of Hangul recognition, a similar 3.5-fold reduction of the error is achieved, improving classification accuracy from 92.0% to 97.9%. Minimal hardware modifications to an existing CGRA and automatic compiler optimizations allow us to effectively accelerate training on the device. Through applying these optimization the CGRA can be used as a low-power CNN accelerator to retrain yielding an average speedup of 36× with a 54-fold energy reduction compared to an ARMv8 processor. A speedup of 42 and a 12-fold reduction in energy consumption is achieved over a 3-way VLIW processor.

In summary, the main contributions of this work is a novel deep neural network architecture that allows for adaption of existing CNN architectures with a small Augmenting Engine that can be re-trained on-device using a small set of user-specific data. Additionally, it is shown that by applying compiler optimizations and minor hardware adaptions to an existing CGRA accelerator, improved CNN training performance can be achieved. In totality, the proposed network structure and the optimizations enable low-power customization of CNNs to a particular user on device.

The remainder of this paper is organized as follows: A simple motivational example is given in Section 2 and the proposed structure is defined in Section 3. Section 4 details the hardware and compiler optimizations that enable a CGRA to be used as a CNN accelerator. Section 5 shows how the proposed design can be applied to the problems of handwritten Latin and Korean character recognition. The experimental setup and evaluation of the approach are given in Sections 6 and 7. A brief overview of related techniques and the conclusions are presented in Section 8 and 9.

2. Motivation

Consider a mobile device such as a smartphone that includes a system to recognize hand-written Latin letters and digits. The system is implemented with a CNN and trained using a large set of general training data. Training of the CNN is performed by the vendor, and testing at the factory yields an accuracy of around 88% [4]. The character-recognition system is then implemented in the device and shipped the end user - this is the prevalent business model of today's mobile device vendors or service providers offering DNN-enabled functionality. However, when the end user applies the system to their own handwriting, it may result a disappointingly low character recognition accuracy, especially if the user has a unique writing style. The reason is that a model trained on general data cannot take into account all possible variations

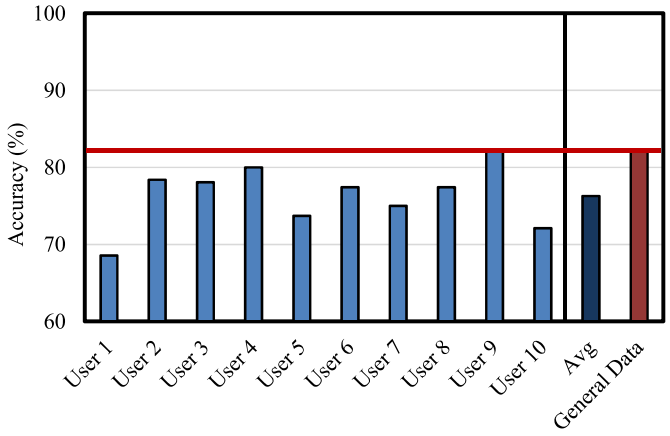


Fig. 1. Accuracy of the general NIST '19 model on user data.

in user writing style. This is especially the case when the problem contains similar classes, such as the characters [I, l, 1]. The way in which a person writes an "l" can be nearly identical to another person's "1". This inherently limits the accuracy of the general model for a particular user.

The problem can be demonstrated with a simple experiment. The LeNet-5 network model [21], modified to recognize 62 classes (26 upper-, 26 lower-case letters, and 10 digits), is trained on the NIST Special Database 19 [20] containing all 62 handwritten letters and digits. The model achieves a test accuracy on the general training set of 82.1%. However, when this model is tested on individual user data, the average recognition accuracy drops to 76.1% (Fig. 1) with one user obtaining less than 70% accuracy. Examples of user data is shown in Fig. 2, demonstrating how writing styles can and do vary between different users.

More sophisticated and larger CNNs trained with more data are expected to achieve higher accuracy on unseen data. These, however, are also harder to train and embed into devices due to their increased complexity, yet the inherent limitations of general models remain. This is often a problem as it is difficult for the training set to cover the exact real life scenario. The model, therefore, needs to adapt to the new user data with its different properties but perform the same task, i.e. recognize the same classes. This is known as the *domain adaption problem* or, specifically for character recognition, as *writer adaption*. This problem of adaption is particularly an issue for human-generated data such as handwritten characters or speech where variations can be unique to each user and differences can vary significantly. What is required in this case is a model based on a powerful general classifier that can be adapted to an individual user with a small amount of overhead in terms of complexity and number of required training examples. Such a system, employed in personal mobile devices, must be able to quickly learn the characteristics of the end user's data on the device itself in order to provide satisfactory recognition accuracy on embedded systems.

3. Methodology

A network architecture overcoming the aforementioned problems is presented herein. The standard network trained on general data is augmented by a small auxiliary network. Together, the networks are able to quickly adapt to a user's specific data. This section describes the network architecture, its initialization, and the user specialization in more detail.

3.1. Initialization

It is assumed that a CNN classifier to classify images into predetermined classes is to be incorporated into a mobile or embedded device.

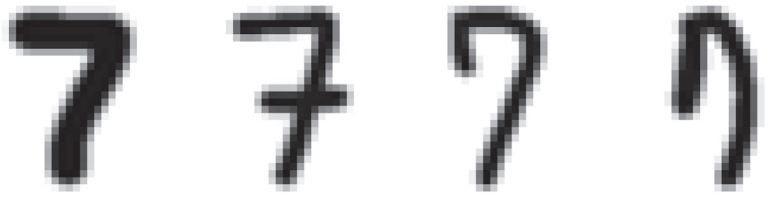


Fig. 2. The character '7' written by four different users.

In one of its simpler instantiations, such a classifier could be used to recognize handwritten characters, in a more complex form this could be used to classify images from a camera feed. Several implementations of such networks exist in the form of either a dedicated hardware accelerator or as a software implementation. We call this system the *basic* inference engine (BIE).

For the purposes of the proposed design, the BIE is a black box 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. This can be trained on either a large publicly available or private general dataset and can be included in the mobile device. For our purposes, the BIE does not have to be re-trainable and could be implemented in software and accelerated using a dedicated accelerator for embedded inference, a mobile GPU, or even a fixed implementation in hardware. Such as system suffers from the problem described previously, namely, that performing well in the general case does not necessarily mean good accuracy for individual users.

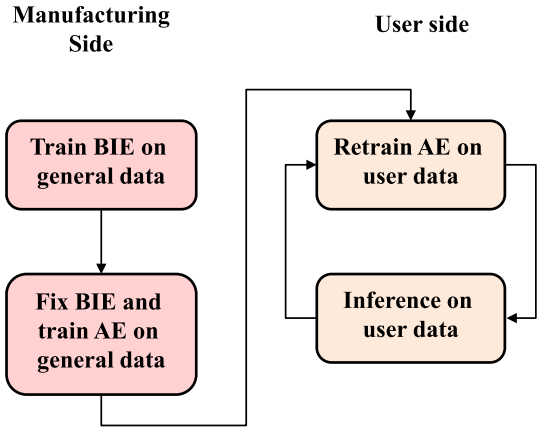


Fig. 4. Training flow.

3.2. On device training

Although there has been much work on accelerating on-device inference there has been little work on developing on-device training accelerators. To enable on-device user specialization, the BIE is augmented with a smaller *augmenting engine* (AE) as shown in Fig. 3. This much smaller network can be executed on an existing general purpose low power processor in parallel to the larger, but highly accelerated network. The AE comprises two parts: a simple convolutional network, labeled C, followed by a fully-connected layer labeled B. The basic idea behind this novel system structure is illustrated in Fig. 4. First, the BIE is trained on a large set of general data or purchased in a trained state, then its weights are fixed. Second, blocks B and C of the AE are attached to the outputs of the BIE. The entire network composed of the BIE and the AE is retrained on the general data to initialize the weights of blocks B and C. These two steps constitute the general training performed by the device manufacturer using general data. The device is received by the end user in this state.

The purpose of block C is to learn the user data, and the function of block B is to learn to combine the outputs of the BIE and the C block adaptively. Instead of feeding the same input data to the BIE and the AE, the activations from a hidden layer of the BIE could be used as inputs of the AE. This allows the AE to re-use the results generated by the general classifier layers at the beginning of the BIE and, at the same time, to reduce the overhead of the AE.

On the user side, the weights of the BIE remain fixed, and only blocks B and C are retrained with the user-specific dataset. By using the classifier to recognize handwritten characters, the user continuously trains the system. This can be done without active user feedback. For an application that converts handwritten notes into text, for example, a language model can be used to suggest word completions similar to those in use in smartphones. When the user selects a suggested word,

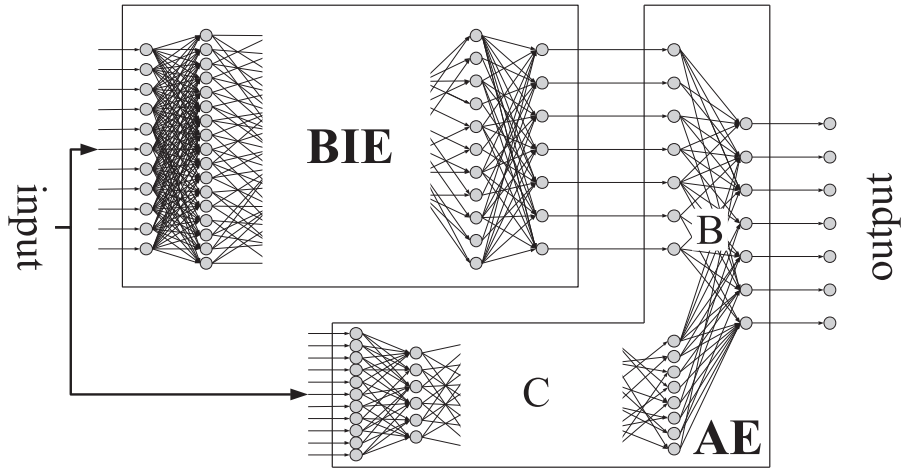


Fig. 3. Augmenting the basic inference engine (BIE) with the augmenting engine (AE).

this information is used to label the handwritten character data and transparently perform supervised learning.

4. Training neural nets with CGRAs

Rather than developing an embedded training accelerator and taking up valuable space with an additional ASIC or FPGA, we propose the use of an existing CGRA processor to accelerate the AE. These are a class of low power general accelerators already in use in several embedded devices. Due to the small size of the AE network, a high powered mobile GPU or neural processor would be an inefficient use of resources. For this particular work we focus on using a CGRA as our AE accelerator. CGRAs have been proposed for and are primarily used to accelerate multimedia workloads in embedded devices [16–19]. This section describes the structure and operation of the CGRA targeted in this work, the Samsung Reconfigurable Processor [15], and the hardware- and software optimizations to increase its performance when processing CNNs.

4.1. Coarse-grained reconfigurable arrays

CGRA processors consist of an array of heterogeneous processing elements (PEs) and register files (RFs). Fast on-chip data storage is provided by SRAM or eDRAM. The class of CGRA accelerators targeted in this work operates in data-flow mode. Loops are converted into a compressed software-pipelined modulo-scheduled representation by a CGRA compiler [22,23]. If the weights and activations all fit into the on-chip memory, expensive off-chip accesses can thus be avoided. While traditionally used for multimedia workloads, CGRAs are well-suited to accelerate CNNs as the latter consist of loops with high iteration counts and sufficient loop-level and instruction-level parallelism that can be exploited by the heterogeneous processing elements of the CGRA.

The starting point of the considered CGRA implementation is the Samsung Reconfigurable processor, a hybrid VLIW/CGRA processor based on the ADRES architecture [24]. Software-pipelinable loops are executed on all PEs of the CGRA, while control flow-intensive code is executed in VLIW mode with only two or three PEs active. The SRP C compiler computes the code for the modulo loop schedules and the control-flow intensive parts, then inserts the necessary glue code.

4.2. Architecture optimizations

In its basic architecture configuration, the SRP is already fairly well-suited to process CNNs. The basic configuration defines a 32-bit floating-point CGRA with 4×4 heterogeneous PEs and a total of 320 KB of on-chip data SRAM. Four PEs are connected to the data memory and support memory instructions while half of the 16 PEs support floating point operations. (Fig. 5). To ensure the SRP can continue to be used as a general-purpose accelerator and not to increase power usage and chip size, modifications to its architecture should be minimal. Therefore, additional processing elements are not considered. Primarily the number and positioning of floating point units along with the amount of on-chip memory is the architectural design space explored. These two areas are chosen due to the specific properties of CNN layers that consist primarily of matrix multiplications of floating point weights with input values to produce an output, meaning that floating point and memory operations dominate the overhead. To accommodate the frequency of these instructions, the following hardware modifications of the SRP are explored. The on-chip SRAM is replaced by 4 MB of 3D-stacked eDRAM [25] comprising of eight banks in a similar layout as proposed by DianNao [10] to reduce the number of off-chip memory accesses. This means that larger AE designs for more complex networks can be accommodated entirely in on-chip memory. On the compute side, four hardware modifications were considered as shown in Fig. 5. These involve increasing the number of memory-enabled PEs, the number of floating point-enabled PEs, or both. In many other CNN accelerators, fused multiply-add (MAC) units are used to speed up matrix multiplication. For CGRAs, however, we found that supporting MACs is not necessarily an effective approach as the loops are software pipelined, meaning the performance limiting factor is not how long a single loop

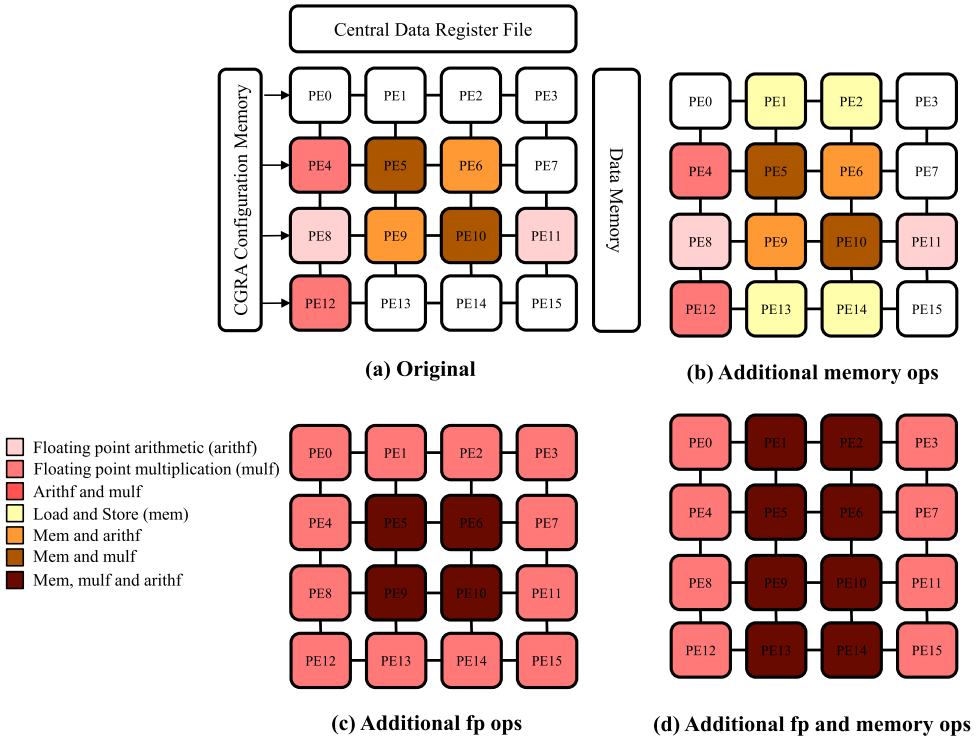


Fig. 5. SRP hardware configuration variations.

Table 1
Overhead and accuracy for different Latin AE designs. The selected configuration is shown in **bold** typeface.

Pooling	Channels	MAC (kOps)	Memory (kB)	Start Accuracy (%)	End Accuracy (%)
No pooling	1	40	112	75.9	93.2
5	169	421	74.9	93.1	
10	330	808	74.4	92.8	
20	653	1582	74.5	92.9	
50	1622	3904	73.4	92.9	
1/2 pooling	1	13	50	75.7	92.9
5	36	104	75.3	93.1	
10	**64**	**172**	**76.1**	**93.7**	
20	120	308	75.7	93.4	
50	289	716	75.6	93.8	
1/4 pooling	1	8	37	75.6	93.1
5	11	46	76.0	92.7	
10	15	58	76.2	93.1	
20	23	81	75.4	93.2	
50	45	149	76.1	93.0	

iteration takes but how frequently a new loop iteration can be started, i.e., the so-called, the *initiation interval* (II) [26].

A *data-memory queue* enqueues, serializes, and dequeues memory requests issued by the PEs to the different eDRAM banks [15]. With more memory-enabled PEs, the probability of bank conflicts increases. To avoid stalls caused by bank conflicts, the latency of the load instruction is increased from 16 to 32 cycles. The additional cycles allow the data-memory queue to buffer memory requests before and after the memory banks in case of conflicts. This increased memory latency has a negligible effect on the CGRA's throughput because the memory latency can be hidden in software-pipelined loops; i.e., the throughput of a scheduled loop is determined by the initiation interval rather than the schedule length of a single loop iteration [26].

4.3. Compiler optimizations

Code executed on the SRP runs bare-metal. 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 like Caffe [27] or Tensorflow [28] depend heavily on libraries. For this reason, a framework written in the C language and without any external dependencies was developed to support embedded accelerators such as CGRAs [29]. The framework supports Caffe-style input files, including networks with multiple branches that are required for implementing the proposed BIE-AE structure. The LLVM-based SRP C compiler takes the C code generated by the framework and creates code capable of executing on the SRP. Therefore, possibilities for code optimization at the compilation stage are also explored.

CNN processing consists primarily matrix multiplications; most accelerators employ some form of a highly-optimized implementation of a general matrix multiplication (GEMM) kernel. Matrix multiplication is composed of fairly simple nested loops with high iteration counts. Such loops are ideal for CGRA processors that exploit parallelism both at the loop and instruction level. In order to achieve the best possible performance, a loop nest should (a) have a sufficiently large innermost loop body that, when software pipelined, leads to a good utilization of the PEs, and (b) have a high iteration count for the innermost loop as only that loop is executed on the CGRA while the control-flow intensive code of outer loops is executed in VLIW mode. An automatic code-optimization framework [30] is employed that explores the large parameter space of the traditional loop optimization techniques such as loop interchange, loop fusion, and loop unrolling. The steps of this

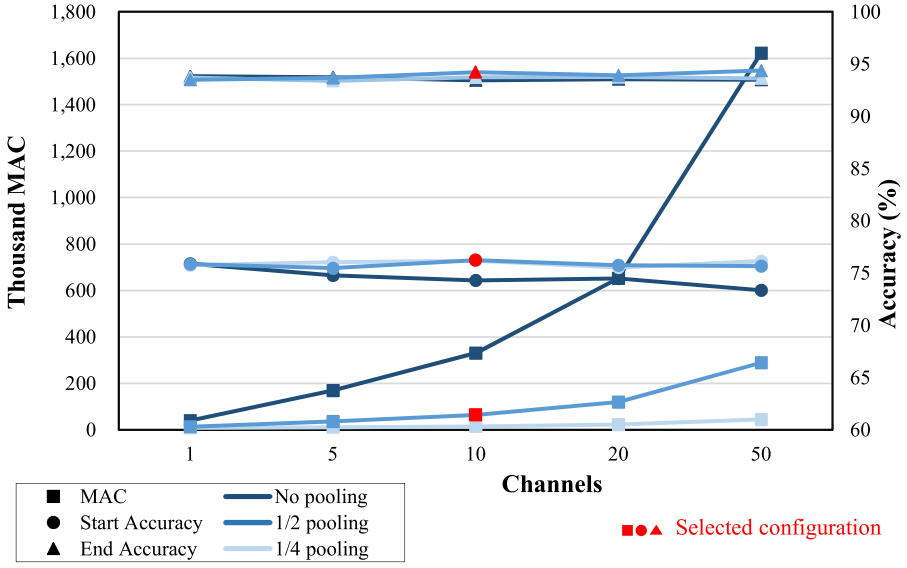


Fig. 6. Overhead and accuracy for different Latin AE designs with the BIE activated.

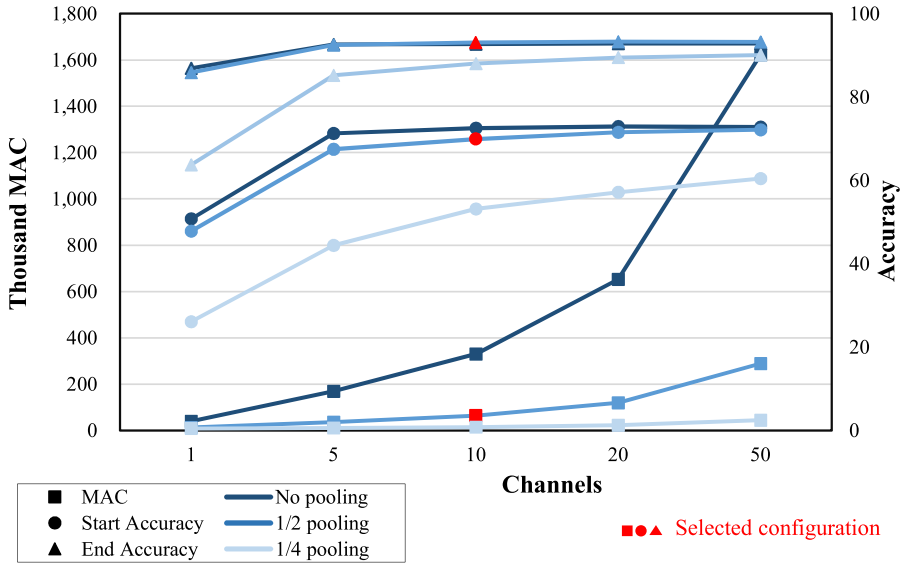


Fig. 7. Overhead and accuracy for different Latin AE standalone designs.

optimization are described in more detail in the following paragraph.

First, the C code is split into separate files for each layer. This allows us to perform aggressive constant propagation of the loop iterations counts from information contained in the network specification. For example, the dimensions of the input, the number of channels, and the size and stepping of the kernel can be propagated into the code. After propagating constants into the loop bounds of the GEMM algorithm, the compiler automatically performs loop unrolling and loop interchange to maximize the instruction-level parallelism and increase the iteration count of the innermost loop. Computing a modulo schedule for a loop can take several minutes, hence the ideal unrolling factors of a loop nest are estimated using a cycle prediction model. The model iterates through each possible combination of unrolling factors for each nesting level and estimates the runtime cycles of unrolling by this factor. After the loop nest has been unrolled, the nested loops are interchanged. The goal of loop interchange is to move the loop with the highest iteration count to the inner-most position. This can be performed on nested loops that have no loop carried dependencies as is the case for GEMM-based loop kernels. The loop transformations are performed along with the existing if-conversion and inlining optimizations [30]. An evaluation of the effect of the different optimizations is discussed in the evaluation section.

5. Design and implementation

The presented design is evaluated with two problems appropriate for user customization. These are the problems of handwritten Latin character classification and the more complex problem of Korean character (Hangul) recognition. For both problems training sets and network descriptions are publicly available.

5.1. Latin handwritten character recognition

For Latin character classification, the NIST '19 dataset [20] is chosen. The dataset consists of a total of 62 classes: lower-case characters "a"-"z", upper-case characters "A"-"Z", and the digits "0"-"9". NIST '19 consists of 731,668 and 82,587 images for training and testing, respectively. For the BIE, a version of LeNet-5 [21] is adapted to produce 62 outputs. Alternative implementations exist, for example, the network described in Ref. [4]; however, this is based on a committee of multiple networks working together in parallel, rendering it less suitable for the embedded domain.

Several factors were considered when choosing the design for the AE. A basic convolutional network consisting of one convolutional layer, one pooling layer, and one fully connected layer is used as a starting point. Two design variables were then investigated, one being the number of channels in the convolutional layer, the second being an average pooling layer at the beginning of the network to downsample the input. Configurations with one, five, 10, 20, and 50 channels were explored. For the downsampling with the pooling layer, the following three settings were tested: no downsampling, downsampling by half, and downsampling to a quarter of the original input size. Table 1 shows how the accuracy and overhead of the AE designs change in dependence of these two parameters, and Fig. 6 visualizes the data from Table 1. The number of MAC operations is proportional to the execution time of the network, and the *Memory* column lists the total memory requirements of the weights and activations used for inference. *Start accuracy* and *end accuracy* list the accuracy of the entire network after training on big data (i.e., before user specialization) and after training on the user's data (i.e., after user specialization).

Table 2
Absolute and relative overhead of the Latin AE with respect to the BIE.

	Training			Inference		
	BIE	AE	Ratio	BIE	AE	Ratio
MAC (kOps)	4376	64	2%	2319	44	2%
Activations (kB)	155	15	10%	79	9	12%
Weights (kB)	3652	157	4%	1826	78	4%

The table and the figure show that downsampling (pooling) has a particularly large effect on the computational overhead. The start accuracy is dominated by the accuracy of the BIE and does not show much variance for the different configurations. This is, somewhat surprisingly, also true for the end accuracy. We also consider the properties of the AE designs in a standalone setting. This is to investigate whether this could be used without the BIE in low-power or extremely resource constrained environments. Fig. 7 shows the start and end accuracy of the AE in such a setting. We observe that in a standalone setting, the number of channels and the downsampling significantly affect both the start and the end accuracy, demonstrating the symbiosis between the BIE and the AE. For the Latin AE, the configuration with 1/2 pooling and 10 channels was selected because it shows the best accuracy at a low computational and space overhead when used with the BIE and in a standalone configuration.

Table 2 lists the absolute and relative overhead of the AE compared to the BIE on a single image using single-precision floating point numbers for weights and activations. Both the training and inference overhead are minimal in comparison with the relatively simple BIE.

Fig. 8 (a) shows the structure of the BIE (LeNet-5) and AE for Latin character recognition. In block C, an average pooling layer is first used to downsample the input to a 14×14 resolution, followed by a convo-

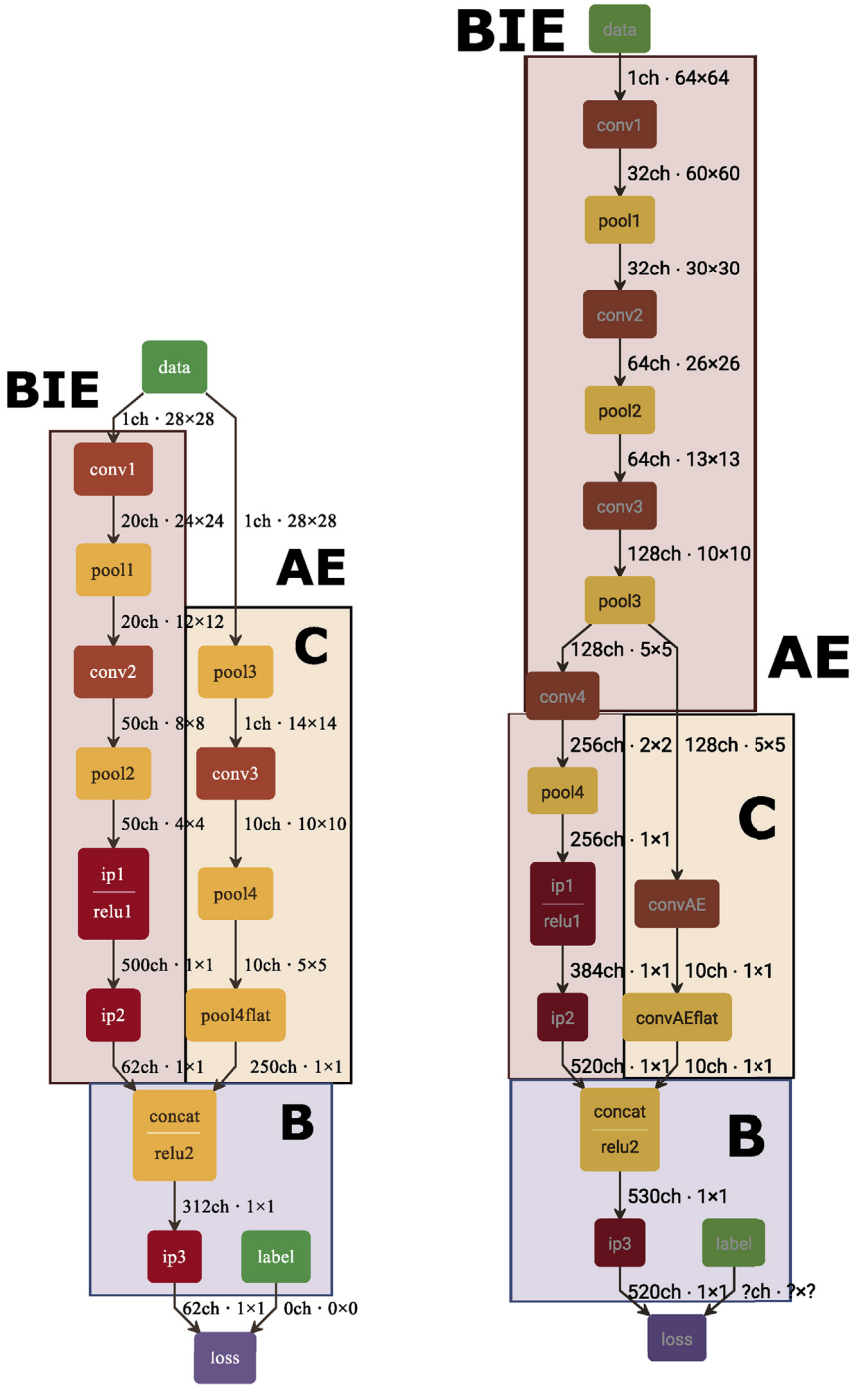


Fig. 8. Latin and Hangul BIE-AE implementations.

Table 3
Hangul AE designs.

Input location	Input dimensions	Channels	MAC (kOps)	Memory (kB)	Start Accuracy (%)	Final Accuracy (%)
Image	64×64×1	1	4375	17,189	90.4	98.2
		5	19,711	77,209	87.5	96.7
		10	38,881	152,234	86.3	95.6
		20	77,221	302,284	67.0	88.8
		50	232,753	765,484	n/a	n/a
Pool 1	30×30×32	1	1785	5107	90.8	98.0
		5	6764	16,402	84.4	95.8
		10	12,987	30,522	85.2	95.3
		20	25,434	58,762	76.1	89.0
		50	62,773	143,481	75.9	89.1
Pool 2	13×13×64	1	756	2561	90.5	97.6
		5	1618	3963	91.0	97.1
		10	2695	5715	89.3	97.5
		20	4850	9219	87.0	97.1
		50	11,313	19,731	86.3	95.8
Pool 3	**5×5×128**	1	548	2210	93.3	98.4
		5	578	2329	91.2	98.5
		10	**615**	**2478**	**92.0**	**97.9**
		20	690	2776	92.1	98.5
		50	913	3669	92.5	98.6

lutional layer with a 5×5 filter size and a 10 channel output. A max pooling layer then further downsamples the outputs to a 5×5 size. The output of the pooling layer is flattened and forwarded to block B where it is attached to the output of the BIE. The combined output is matched to the 62 classes by a fully connected layer.

5.2. Korean handwritten character recognition

The Korean alphabet *Hangul* is an alphabet where about 40 individual characters are combined into a composite character with a total of 11,172 valid characters combinations. The vast majority of these combinations, however, are not used day to day. For Hangul character recognition, the SERI95a database is used. This database consists of the 520 most commonly used handwritten Korean characters, with approximately 1000 examples for each character. The network described in Ref. [31] is used as the BIE; this network achieves state-of-the-art accuracy on the SERI95a task. Additionally, an FPGA implementation of this network exists [32]. This, if implemented as a dedicated hardware chip, would make a good component in the BIE-AE architecture.

Several AE designs are again considered. The starting point is a simple convolutional network, this time consisting of a convolutional layer and a fully connected layer. In this design, the critical overhead is the memory used to train the AE. This is due to the large number of weights required for a fully connected layer with 520 outputs. It is key that any output from the convolutional layer of the AE does not add too many outputs before the final fully connected layer, as each additional value will add 520 weights. For this reason, rather than investigating downsampling, here using an input from an intermediate layer of the BIE is investigated. This allows the AE to use much smaller inputs, resulting in a smaller output with the additional advantage of reusing the features discovered earlier in the BIE network. The two variables investigated for this design are the input layer of the BIE that serves as the input to the AE and the number of channels used in the convolutional layer.

In Fig. 8 (b), the possible locations in the BIE serving as inputs to the AE are the original image and after pool1, pool2, and pool3. The fourth layer is not considered because its output dimension of 1×1 pixel does not allow for further convolutions, making it difficult to capture new spatial features when retraining on user data. The investigated configurations are listed in Table 3. The accuracy is low for configurations with inputs from the early layers of the network; this is due to the extremely large number of weights resulting in a model that is difficult to train with a limited number of user data points. The design using the input raw image with 50 convolutional channels was unable to be trained even on a GPU due to its high memory demands.

For the Hangul AE, the selected configuration uses the 5×5×128 output from the third pooling layer and 10 channels. This configuration has a moderate overhead in terms of MAC operations and memory requirements, but achieves satisfactory accuracy before and after user specialization. The convolutional layer in block C in Fig. 8 (b) uses a 5×5 kernel that reduces the input to a 1×1 output, thereby removing the need for a pooling layer. The flattened output is merged with output of the BIE, followed by a fully-connected layer that produces the final 520 output classes.

Table 4 lists the absolute and relative overhead of the AE relative to the BIE and for one image using single-precision floating point numbers for weights and activations. The memory overhead of the design is not one that can be reduced easily as 520 outputs are necessary for the network to function, however, the chosen design still fits in the on-chip eDRAM memory of the proposed CGRA configuration.

6. Experimental setup

The proposed BIE-AE networks are trained with a publicly available general dataset. To test the user specialization with data from individual users, we implemented an Android application that was then used to collect the user datasets. For the experiments on the Latin alphabet, 40

Table 4
Absolute and relative overhead of the Hangul AE with respect to the BIE.

	Training			Inference		
	BIE	AE	Ratio	BIE	AE	Ratio
MAC (kOps)	103,814	615	0.6%	52,994	308	0.6%
Activations (kB)	1746	17	1.0%	881	15	1.7%
Weights (kB)	8042	2461	31.0%	4021	1230	31.0%

Table 5
Overview of compared processor architectures.

Architecture	Type	PEs	Clock (MHz)	Power (mW)	Technology (nm)
ARMv8-A [35]	general-purpose CPU	4	1200	271	20
VLIW [15]	general-purpose accelerator	3	500	50	32
CGRA	general-purpose accelerator	16	500	150	32
Jetson TX2 [36] [a]	mobile GPU	256	850	4800	20

[a] The Jetson TX2 is operated in its most energy-efficient configuration (Max-Q).

samples for each of the 62 alphanumeric classes were collected from ten individual users. Of the 40 sets of characters from each user, 30 are used to train the AE and the remaining 10 sets form the test set. For the Hangul experiments, 20 sets of the 50 most commonly used characters were collected from seven individual users. 10 sets are used for training, the other half comprises the test set.

Both the general dataset and the user dataset are pre-processed in the same fashion with the techniques used for preprocessing the EMNIST dataset [33]. First, characters have Gaussian smoothing applied, are centered, padded, and scaled down to the relevant input size of the network. For the NIST network this is 28×28 pixels whereas the Hangul network uses 64×64 pixels. The networks are first trained with the relevant general dataset, subsequently the BIE is fixed, then the AE is retrained with the user-specific data one set at a time.

The target CGRA comprises a total of 16 32-bit PEs with 8–16 PEs capable of executing floating point operations and 4–8 PEs supporting memory operations. The different configurations are listed in Fig. 5. The processor is manufactured with 32 nm technology and runs at 500 MHz. The on-chip 3D-stacked eDRAM has a size of 4 MB. The power consumption of the CGRA base array is given by the manufacturer, the power of the eDRAM is modeled using Destiny [34]. The compiler optimizations are implemented into the SRP C compiler. A custom DNN framework is used to generate the C code equivalent of the AE network which is then automatically optimized using the compiler framework as described in Section 4.3. The generated code is executed on a cycle-accurate SRP system simulator in various hardware configurations. Since the CGRA is statically scheduled and runs at a fixed frequency, the runtime can be reliably calculated with the cycle-accurate simulator.

The results of the CGRA are compared against three other architectures often found in embedded systems: a general-purpose ARMv8-A processor [35], the VLIW part of the SRP [15], and Jetson TX2 [36], a mobile GPU processor optimized for deep learning. Measurements for the ARMv8 and the Jetson are executed on real systems; for the CGRA and VLIW results, a cycle-accurate simulator is used that simulates the core and the memory. Energy is computed by multiplying the average power consumption from Table 5 with the runtime of the different architectures.

7. Evaluation

This section evaluates the performance of the presented BIE-AE network structure with respect to user specialization and discusses the effect of the hardware and compiler optimizations.

7.1. Handwritten character recognition of the Latin alphabet

The Latin BIE-AE network from Fig. 8 (a) is trained using the general-purpose dataset, then iteratively retrained using data from an individual user. Fig. 9 shows the gain in accuracy for individual users as the AE is retrained on a user datasets. Each training set consists of one image from each class and is trained on for ten epochs. The initial accuracy after training with the general dataset falls below 70%, but we observe a rapid improvement of the accuracy with an increasing

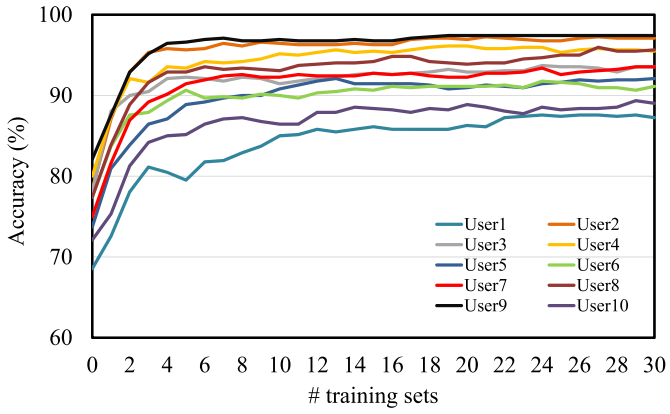


Fig. 9. Accuracy of BIE + AE on Latin user data.

number of training sets. After 30 training sets, the average accuracy has improved from 76.1% to 93.7%.

Table 6 lists the results before and after training for each of the five categories of NIST. For all categories a significant increase in accuracy is achieved, often surpassing the accuracy of the general model significantly. This demonstrates that the BIE-AE model can successfully adapt to a user's writing style.

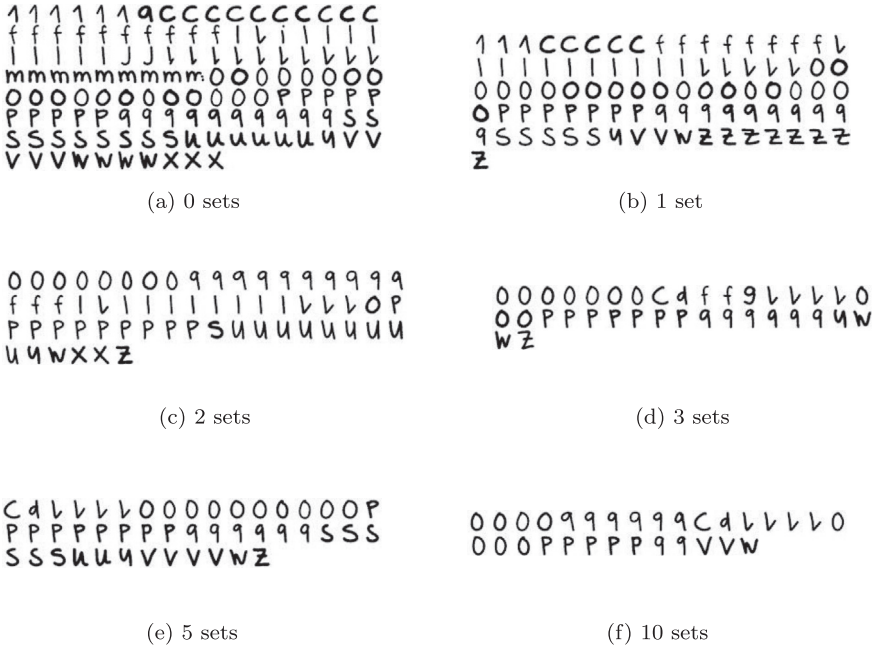
Much of the errors in this problem domain stem from misclassified similarly looking characters. Confused are predominantly characters where the lower and upper case bear resemblance (such as f, F) or similarly shaped characters (i,l,1,j). For an individual user, there is usually some consistency in the way these characters are written. While a general network has no possibility to learn and adapt, the presented BIE-AE model successfully adapts to an individual user's writing style, demonstrating the effectiveness of user customization.

Fig. 10 illustrates how the BIE-AE structure learns to adapt to a user's writing style. The subfigures (a) to (f) show the misclassified characters after training with 0, 1, 2, 3, 5, and 10 sets of user data. We observe that the way the network improves is somewhat systematic. Fig. 10 (a) lists all characters that the general model failed to classify correctly. After one training set, the misclassified m's all become correctly classified as well as many of the x, c, u, w, v, p, and 1 characters. For subsequent steps, improvements are more concentrated in certain areas. After the second training set, the system learns how to distinguish between upper and lower case o's, however, as a result there is an increase in 0's being misclassified. A similar pattern can be observed between q's and 9's. These difficult to distinguish characters eventually become more accurately classified, as can be seen in Fig. 10 (f) after 10 training sets where 0,o and Os along with 9 and q are better classified and the balance of misclassification is more evenly spread between the classes.

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. Fig. 7 shows that the initial accuracy of this AE-only general model is lower, with an average starting 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,

Table 6
Accuracy before/after retraining the Latin BIE-AE model using user-specific data.

Dataset	all		letter		lower		upper		digit	
	before	after	before	after	before	after	before	after	before	after
User 1	67.7	87.6	74.6	91.2	86.5	96.2	95.8	98.5	97.0	99.0
User 2	76.9	97.6	78.1	98.7	94.2	100	100	100	91.0	100
User 3	77.7	93.9	76.0	94.8	97.3	98.9	99.2	99.6	98.0	100
User 4	78.5	96.3	78.5	96.5	95.4	98.9	99.2	100	99.0	100
User 5	76.1	93.2	73.3	92.3	85.4	98.9	94.2	98.5	98.0	100
User 6	77.6	91.5	79.2	93.3	97.3	99.2	97.7	100	99.0	100
User 7	75.3	93.9	75.6	96.5	90.8	99.6	100	100	100	100
User 8	77.1	95.3	78.7	96.5	93.1	99.6	98.5	100	100	100
User 9	82.1	97.9	81.2	97.9	95.0	99.2	99.6	100	100	100
User 10	72.3	89.5	72.7	90.4	90.4	98.8	89.6	97.7	93.0	99.0
Average	76.1	**93.7**	76.8	**94.8**	92.5	**98.9**	97.4	**99.4**	97.5	**99.8**



(a) 0 sets (b) 1 set

(c) 2 sets (d) 3 sets

(e) 5 sets (f) 10 sets

Fig. 10. Misclassified images from test set while retraining.

which is particularly relevant for the user customization scenario, as for end users it is important to have as high accuracy initially and improve this as quickly as possible. Since the accuracy of the standalone personalized AE is still higher than that of the BIE trained on general data, a standalone configuration may make sense for low-power environments.

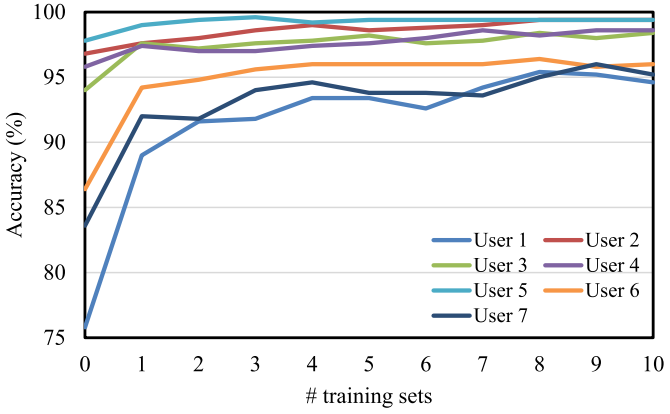


Fig. 11. Accuracy of BIE + AE on Hangul user data.

7.2. Korean handwritten character recognition

To recognize Korean handwritten characters, the network show in Fig. 8 (b) is used. Fig. 11 shows similar improvements as for Latin characters for individual users with an increasing amount of user specialization. Each training set consists of one character from each collected class and is trained on for 100 epochs to increased the rate of learning. The average starting accuracy on the user data is considerably higher than for Latin characters at 92.0%, but still improves significantly to

Table 7
Accuracy before/after retraining the Korean BIE-AE model using user-specific data.

Dataset	before	after
User 1	81.5	94.4
User 2	96.6	98.8
User 3	94.6	99.0
User 4	95.0	98.4
User 5	99.0	99.8
User 6	89.9	98.0
User 7	87.5	96.8
Average	92.0	**97.9**

Table 8
Comparison of different hardware configurations.

Fig. 5	(a) original	(b) add mem	(c) add fp	(d) add mem & fp
Speedup	33.9	46.1	37.6	46.7

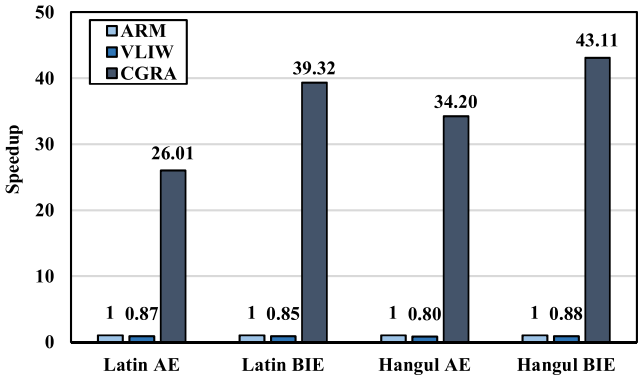


Fig. 12. CGRA optimization compared to ARMv8.

97.9% as additional user data is added. The detailed numbers for each individual user are listed in Table 7.

7.3. Hardware optimizations

The four hardware configurations shown in Fig. 5 are evaluated using the BIE design for the NIST network. This is chosen rather than the AE as for larger CNNs optimizations should become more apparent. The performance in terms of speedup compared to running the BIE on the VLIW portion of the hybrid processor is shown in Table 8. The automatic compiler optimizations are enabled for all architectures as without them the underlying CGRA architecture would require manual optimizations to achieve satisfactory performance (an evaluation of the different compiler optimization is given in the following section). The results show that the addition of the additional memory units yields the greatest increase in performance, while additional floating point units improve performance only marginally. In the following, all results were obtained using architecture (b) that, even though a bit slower than design (d), is closer to the original SRP and less specialized, thus better suited for general-purpose acceleration.

7.4. Compiler optimizations

The modified SRP is compared to an ARMv8-A processor, a 3-issue VLIW, and the Jetson TX2 mobile GPU. Fig. 12 shows the speedup relative to the ARMv8 architecture. On average, a speedup of 36× is measured relative to ARMv8 and 42× relative to the VLIW for four CNNs comprising the Latin character recognition AE and BIE and the Hangul character recognition AE and BIE. The larger BIEs profit more from the compiler optimizations because of the higher innermost loop-iteration counts. The larger BIEs do, of course, have a greater absolute overhead compared to the AEs as can be seen in the processing times of the tested CNNs shown in Table 9.

Table 9 also gives an indication whether the performance of the CGRA is sufficient in order to not slow down the faster BIE. Assuming a state-of-the-art DNN accelerator such as the Jetson TX2 is used for the BIE, we observe that the processing times of inference and training on the BIE for both alphabets are always higher than that of the AE on the CGRA. Since the BIE and the AE cannot run completely in parallel (Fig. 8), the latency of a single operation will be slightly higher, however, for batch processing the AE does not slow down the BIE. For Hangul inference, for example, the BIE on the Jetson TX2 takes 1 ms to execute, but the results of the AE on the CGRA are available after 0.7 ms. We also observe that large dedicated CNN accelerators such as the Jetson TX2 do not perform well on small networks like the AE. Despite its computing power, inference and training of the AE is significantly faster on the CGRA for both presented AEs.

Finally, a comparison of the energy consumption of inference and training on one image is given in Table 10. With the Latin AE, energy is reduced 47 times compared to the ARM processor. For the larger Hangul AE, the energy consumption is reduced 62-fold. Compared to the less powerful VLIW, the hybrid CGRA still achieves a 10-fold reduction in energy for the Latin AE and 14-fold reduction for the Hangul AE. The Jetson TX2 consumes several orders of magnitude more energy than the SRP; this is both caused by the high average power consumption and the relatively bad performance on small CNN networks.

Table 10
Energy consumption of inference and training on one image.

Architecture	Latin AE Energy [mJ]		Hangul AE Energy [mJ]	
	infer	train	infer	train
ARMv8	0.80	3.27	6.70	24.55
VLIW	0.14	0.70	1.30	5.66
CGRA	0.02	0.07	0.10	0.40
Jetson TX2	3.20	12.10	5.10	21.40

8. Related work

The proposed approach to user customization is an example of the domain adaption problem, similar to transfer learning. This addressed using a supervised learning based approach. Several other approaches to domain adaption have been proposed, however this one particularly targets embedded environments. Other approaches to domain adaption include Adversarial Discriminative Domain Adaption (ADDA) [37], this is an unsupervised approach that uses two separate networks for the source and target domains, (general and user data) and a discrimina-

Table 9
Processing time of BIE and AE on different architectures.

Architecture	Processing time (ms)							
	Latin AE		Latin BIE		Hangul AE		Hangul BIE	
	infer	train	infer	train	infer	train	infer	train
ARMv8	3	12	118	489	25	91	2739	10370
VLIW	2.8	14	131	573	26	113	2966	11730
CGRA	0.1	0.46	2.8	12	0.7	2.6	63	240
Jetson TX2	0.7	2.7	0.7	4.8	1.1	4.5	1	8.4

tor network to distinguish between the two. In ADDA a source image is fed into the source network and a target image is fed into the target network, the target network then attempts to map the target images to the source feature space while the discriminator attempts to distinguish outputs of the source and target networks. This adversarial training means that eventually the target network will be able to transform the target images into representations similar to the source network. However, this process requires both general and user data to be available when training the target network and has a large network structure rendering it unsuitable for the embedded domain.

Other ways of adapting existing networks to a new task are the transfer learning techniques of Joint Training [38] and Fine-Tuning [39]. In joint training, the entire CNN is retrained with both general and user datasets to optimize it for both tasks, again unsuitable for the embedded domain due to storage limitations. Fine tuning is where either the whole or part of a pretrained network is adapted to a new task by training either all or part of the network with a reduced learning rate, however this is not possible to perform with a hardware implementation of an existing network, which is an advantage of the proposed structure.

There have been numerous works regarding the acceleration of DNNs on embedded mobile devices. These often make use of FPGAs and ASICs, but concentrate almost exclusively on the inference task. There have been FPGA implementations developed specifically for accelerating certain networks, such as SqueezeNet [40] and the base network used to recognize Hangul in this paper [32]. The proposed choice of the CGRA is to make use of a low power accelerator already present in mobile devices for general purpose acceleration, primarily for the training task.

Another architecture for accelerating DNNs on mobile devices is Eyeriss [41] 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 [42] 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 on embedded devices.

9. Conclusion

A novel CNN-based network structure comprising a base inference engine (BIE) and an augmenting engine (AE) is presented that allows for user customization of CNNs trained on general data. After the initial training, only the smaller AE is retrained, thereby retaining general accuracy yet adapting to the individual user's characteristics with very little overhead. The presented approach is applied to the problem of handwritten character recognition for the Latin and the Korean alphabet. After user specialization, the presented BIE-AE network has a 3.5-fold lower classification error than the standalone BIE, increasing classification accuracy from 76.1 to 93.7% on the Latin alphabet and from 92.0 to 97.9% for the Korean character set. Mapping the AE to modulo-scheduled CGRAs, an existing class of accelerators present in many commercial embedded systems, shows that, after applying automatic compiler optimizations, the CGRA outperforms an ARMv8 and a 3-issue VLIW processor by a factor of 36 and 42 for performance at a 54- and 12-fold reduced energy consumption, respectively.

Acknowledgments

We thank the anonymous reviewers for the helpful comments and suggestions. This work was supported, in part, by BK21 Plus for Pioneers in Innovative Computing (Dept. of Computer Science and Engineering, SNU) funded by the National Research Foundation (NRF) of Korea (Grant 21A20151113068), the Basic Science Research Program through NRF funded by the Ministry of Science, ICT and Future Planning (Grants NRF-2015K1A3A1A14021288 and 2016R1A2B4009193), by the Promising-Pioneering Researcher Program through Seoul National University in 2015, and academic-industrial collaboration funded by Samsung Research, Seoul, Korea. ICT at Seoul National University provided research facilities for this study.

References

[1] B. Harris, M.S. Moghaddam, D. Kang, I. Bae, E. Kim, H. Min, H. Cho, S. Kim, B. Egger, S. Ha, K. Choi, Architectures and algorithms for user customization of cnns, in: 2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC), 2018, pp. 540–547, https://doi.org/10.1109/ASPDAC.2018.8297379.

[2] D. Ciregan, U. Meier, J. Schmidhuber, Multi-column deep neural networks for image classification, in: 2012 IEEE Conference on Computer Vision and Pattern Recognition, 2012, 3642–3649.

[3] A.S. Razavian, H. Azizpour, J. Sullivan, S. Carlsson, CNN features off-the-shelf: an astounding baseline for recognition, CoRR abs/1403.6382. http://arxiv.org/abs/1403.6382.

[4] D.C. Ciresan, U. Meier, L.M. Gambardella, J. Schmidhuber, Convolutional neural network committees for handwritten character classification, in: 2011 International Conference on Document Analysis and Recognition, 2011, pp. 1135–1139.

[5] I.-J. Kim, X. Xie, Handwritten hangul recognition using deep convolutional neural networks, Int. J. Doc. Anal. Recogn. (IJDAR) 18 (1) (2015) 1–13, https://doi.org/10.1007/s10032-014-0229-4.

[6] G. Hinton, L. Deng, D. Yu, G.E. Dahl, A. r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T.N. Sainath, B. Kingsbury, Deep neural networks for acoustic modeling in speech recognition: the shared views of four research groups, IEEE Signal Process. Mag. 29 (6) (2012) 82–97, https://doi.org/10.1109/MSP.2012.2205597.

[7] A. Graves, A. r. Mohamed, G. Hinton, Speech recognition with deep recurrent neural networks, in: 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, 2013, pp. 6645–6649, https://doi.org/10.1109/ICASSP.2013.6638947.

[8] D. Ciresan, A. Giusti, L.M. Gambardella, J. Schmidhuber, Deep neural networks segment neuronal membranes in electron microscopy images, in: F. Pereira, C.J.C. Burges, L. Bottou, K.Q. Weinberger (Eds.), Advances in Neural Information Processing Systems 25, Curran Associates, Inc., 2012, pp. 2843–2851.

[9] Y. Taigman, M. Yang, M. Ranzato, L. Wolf, Deepface: closing the gap to human-level performance in face verification, in: 2014 IEEE Conference on Computer Vision and Pattern Recognition, 2014, pp. 1701–1708, https://doi.org/10.1109/CVPR.2014.220.

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

[11] Y.-H. Chen, J. Emer, 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, IEEE Press, 2016, pp. 367–379.

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

[13] S. Teerapittayanon, B. McDanel, H.T. Kung, Distributed deep neural networks over the cloud, the edge and end devices, in: 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS), 2017, pp. 328–339, https://doi.org/10.1109/ICDCS.2017.226.

[14] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, 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, ACM, 2015, pp. 161–170.

[15] D. Suh, K. Kwon, S. Kim, S. Ryu, 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), 2012, pp. 67–70.

[16] Y. Park, H. Park, S. Mahlke, Cgra express: accelerating execution using dynamic operation fusion, in: International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES), CASES '09, ACM, New York, NY, USA, 2009, pp. 271–280.

[17] C. Kim, M. Chung, Y. Cho, M. Konijnenburg, S. Ryu, J. Kim, ULP-SRP: ultra low power samsung reconfigurable processor for biomedical applications, in: International Conference on Field-programmable Technology (FPT), IEEE, 2012, pp. 329–334.

[18] W.-J. Lee, S.-H. Lee, J.-H. Nah, J.-W. Kim, Y. Shin, J. Lee, S.-Y. Jung, SGRT: a scalable mobile gpu architecture based on ray tracing, in: ACM SIGGRAPH 2012 Posters, ACM, 2012, p. 44.

[19] J. Lee, Y. Shin, W.-J. Lee, S. Ryu, K. Jeongwook, Real-time ray tracing on coarse-grained reconfigurable processor, in: International Conference on Field-programmable Technology (FPT), 2013, pp. 192–197.

[20] P.J. Grother, NIST Special Database 19 Handprinted Forms and Characters Database, National Institute of Standards and Technology.

[21] Y. Lecun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proc. IEEE 86 (11) (1998) 2278–2324.
[22] T. Oh, B. Egger, H. Park, S. Mahlke, Recurrence cycle aware modulo scheduling for coarse-grained reconfigurable architectures, SIGPLAN Notes 44 (7) (2009) 21–30.
[23] H. Lee, M.S. Moghaddam, D. Suh, B. Egger, Improving energy efficiency of coarse-grain reconfigurable arrays through modulo schedule compression/decompression, ACM Trans. Archit. Code Optim. 15 (1) (2018) 1:1–1:26, https://doi.org/10.1145/3162018.
[24] B. Mei, B. Sutter, T. Aa, M. Wouters, A. Kanstein, S. Dupont, Implementation of a coarse-grained reconfigurable media processor for AVC decoder, J. Signal Process. Syst. 51 (3) (2008) 225–243.
[25] R.E. Matick, S.E. Schuster, Logic-based edram: origins and rationale for use, IBM J. Res. Dev. 49 (1) (2005) 145–165, https://doi.org/10.1147/rd.491.0145.
[26] B.R. Rau, Iterative modulo scheduling: an algorithm for software pipelining loops, in: Proceedings of the 27th Annual International Symposium on Microarchitecture, MICRO 27, ACM, New York, NY, USA, 1994, pp. 63–74, https://doi.org/10.1145/192724.192731.
[27] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, T. Darrell, Caffe: Convolutional Architecture for Fast Feature Embedding, arxiv:1408.5093.
[28] TensorFlow v1.0, 2017, https://www.tensorflow.org/.
[29] D. Kang, E. Kim, I. Bae, B. Egger, S. Ha, C-GOOD: C-code generation framework for optimized on-device deep learning, in: Proceedings of the 37th International Conference on Computer-aided Design, ICCAD '18, 2018.
[30] I. Bae, B. Harris, H. Min, B. Egger, Auto-tuning CNNs for coarse-grained reconfigurable array-based accelerators, IEEE Trans. Comput. Aided Des. Integrated Circ. Syst. 37 (10) (2018) 1, https://doi.org/10.1109/TCAD.2018.2857278.
[31] I.-J. Kim, X. Xie, Handwritten hangul recognition using deep convolutional neural networks, Int. J. Doc. Anal. Recogn. (IJDAR) 18 (2014) 1–13.
[32] H. Park, C. Lee, H. Lee, Y. Yoo, Y. Park, I. Kim, 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, 2017https://doi.org/10.1145/3125501.3125522.
[33] G. Cohen, S. Afshar, J. Tapson, A. van Schaik, EMNIST: an Extension of MNIST to Handwritten Letters, CoRR abs/1702.05373. arxiv:1702.05373.
[34] M. Poremba, S. Mittal, D. Li, J.S. Vetter, Y. Xie, Destiny: a tool for modeling emerging 3d nvm and edram caches, in: 2015 Design, Automation Test in Europe Conference Exhibition (DATE), 2015, pp. 1543–1546, https://doi.org/10.7873/DATE.2015.0733.
[35] Arm Limited, Arm Processor Data Sheets, 2017, https://developer.arm.com/products/processors/.
[36] Nvidia Jetson Tx2 Delivers Twice the Intelligence to the Edge, Sep 2018, https://devblogs.nvidia.com/jetson-tx2-delivers-twice-intelligence-edge/.
[37] E. Tzeng, J. Hoffman, K. Saenko, T. Darrell, Adversarial Discriminative Domain Adaptation, CoRR abs/1702.05464. arxiv:1702.05464.
[38] R. Caruana, Multitask learning, Machine Learn. 28 (1) (1997) 41–75.
[39] R. B. Girshick, J. Donahue, T. Darrell, J. Malik, Rich feature hierarchies for accurate object detection and semantic segmentation, CoRR abs/1311.2524. http://arxiv.org/abs/1311.2524.
[40] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, K. Keutzer, SqueezeNet: Alexnet-level Accuracy with 50x Fewer Parameters and <1mb Model Size, CoRR abs/1602.07360. arxiv:1602.07360.
[41] Y.-H. Chen, J. Emer, V. Sze, Eyeriss: a spatial architecture for energy-efficient dataflow for convolutional neural networks, SIGARCH Comput. Archit. News 44 (3) (2016) 367–379.
[42] M. Tanomoto, S. Takamaeda-Yamazaki, J. Yao, 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, 2015, pp. 73–80.