M.S. THESIS

On-Device Personalization of Deep Neural Networks

기기 상에서의 심층 신경망 개인화 방법

FEBRUARY 2019

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
COLLEGE OF ENGINEERING
SEOUL NATIONAL UNIVERSITY

Barend Thomas Harris

M.S. THESIS

On-Device Personalization of Deep Neural Networks

기기 상에서의 심층 신경망 개인화 방법

FEBRUARY 2019

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
COLLEGE OF ENGINEERING
SEOUL NATIONAL UNIVERSITY

Barend Thomas Harris

On-Device Personalization of Deep Neural Networks

기기 상에서의 심층 신경망 개인화 방법

지도교수 Bernhard Egger

이 논문을 공학석사 학위논문으로 제출함

2018 년 10 월

서울대학교 대학원

컴퓨터 공학부

바렌드 토마스

바렌드 토마스의 공학석사 학위논문을 인준함

2018 년 12 월

Abstract

There exist several deep neural network (DNN) architectures suitable for embedded inference, however little work has focused on training neural networks on-device. User customization of DNNs is desirable due to the difficulty of collecting a training set representative of real world scenarios. Additionally, inter-user variation means that a general model has a limitation on its achievable accuracy. In this thesis, a DNN architecture that allows for low power on-device user customization is proposed. This approach is applied to handwritten character recognition of both the Latin and the Korean alphabets. Experiments show a 3.5-fold reduction of the prediction error after user customization for both alphabets compared to a DNN trained with general data. This architecture is additionally evaluated using a number of embedded processors demonstrating its practical application.

Keywords: Deep Learning, On-Device Learning, User Adaptation

Student Number: 2016-25821

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Deep Neural Networks (DNNs) have achieved great success in previously difficult to solve problem domains. Examples include Computer Vision [7, 36], handwriting recognition [8, 26], and speech recognition [18, 11]. These applications are increasingly moving to the embedded domain, leading to research into reducing the size and overhead of DNNs to run on embedded devices [15] along with developing dedicated hardware DNN accelerators [4, 5, 14]. Research has primarily been limited to the inference task, i.e. using a model pre-trained on big data to classify an unknown input. This thesis, however considers on-device training. Here, situations where this can be useful are explored along with methods to allow efficient on-device training.

Training DNNs to become proficient at a task typically requires a lot of training data, for example when training a handwriting classifier, samples from many different users are gathered. The DNN is then trained on these large number of samples on a server using one or more GPUs in order to create a general model. However, these general models have some limitations as they assume that the initial training data can be abstracted to to cover each real world scenario. Realistically, however, it is difficult to obtain training data representative of all real world scenarios. For example, an image classifier can struggle where lighting conditions in the real world differ from training data, or alternatively a handwriting classifier may struggle to classify a particularly unique style of handwriting. This is known as the domain adaptation problem [41].

To allow for domain adaptation data must be collected from the real world setting and used to adapt the DNN to the real world scenario, this is typically performed by retraining part or all of the model. Offloading domain adaptation to the server is possible, however on-device adaptation is desirable for a number of reasons. First, as the embedded domain is considered, connection to a server is not always guaranteed. Second, as user data is required, privacy becomes a concern, particularly in the case of images or speech data. Third, in terms of logistics, when offloading training to the server a separate personalized model for each user must be kept on the server and synchronized to the device for on-device inference. This could cause a large overhead in terms of both network traffic and server storage, due to the size of the model. Retraining a model on-device has some of its own issues, energy and processing power are more of a concern. Existing on-device inference accelerators are not necessarily well suited to on-device training, and often have relatively high power consumption. For this reason a DNN architecture is proposed that enables low-power on-device personalization of a large general purpose DNN. This attempts to exploit existing high powered inference accelerators to accelerate inference of the general model, which we call the Basic Inference Engine (BIE). The BIE is augmented with a smaller network called the Augmenting Engine (AE) which is retrained on user data. This is accelerated using an existing lower powered on-device processor. In totality we call this the BIE-AE architecture.

Although this architecture is flexible, we specifically investigate using the Samsung Reconfigurable Processor (SRP) [38] to accelerate retraining. This is a general purpose Coarse Grained Reconfigurable Array (CGRA) based embedded processor already integrated into several devices [35, 24, 30, 29].

In summary, the main contribution of this work is a DNN architecture that allows for adaption of existing DNNs with a small augmenting network that can be re-trained on-device using a small set of user-specific data. It is also shown that an existing CGRA can be used to achieve reasonable training performance through applying minor hardware adaptations and compiler optimizations. In combination, this allows the low-powered customization of DNNs on-device. This thesis is primarily based on the journal paper [16] which is in turn an expansion of the conference paper [17].

This thesis is organized as follows: First, a simple motivational example is given in Chapter 2, and a background on DNNs and on-device acceleration is given in Chapter 3. The proposed DNN architecture is defined in Chapter 4. This is applied to the problem of Latin and Korean handwriting recognition in Chapter 5. Acceleration on-device is discussed in Chapter 6, specifically how the SRP can be adapted to DNN training. A description of the experimental setup and the results are given in Chapter 7 and 8. An overview of related techniques is presented in Chapter 9 and finally conclusions are given in Chapter 10.

Chapter 2

Motivation

A simple example is shown to motivate the necessity of user/domain adaptation. Consider a DNN designed to recognize handwritten Latin letters and digits. This is trained on a general training dataset consisting of data from several different users achieving an accuracy of around 88% on its test set [8]. This is deployed to user devices and is used to classify user handwriting. However, for certain users the system may achieve a disappointingly low accuracy, particularly if the user has a unique writing style. This is especially the case when the classification problem contains similar classes, in this case the characters [I, l, 1, i]. 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.

A simple experiment was conducted to demonstrate this. The LeNet-5 network model [28], is modified to recognize 62 classes (26 upper-, 26 lower-case letters, and 10 digits), and is trained on NIST Special Database 19 [12], containing all 62 handwritten Latin letters and digits. This model achieves an accuracy

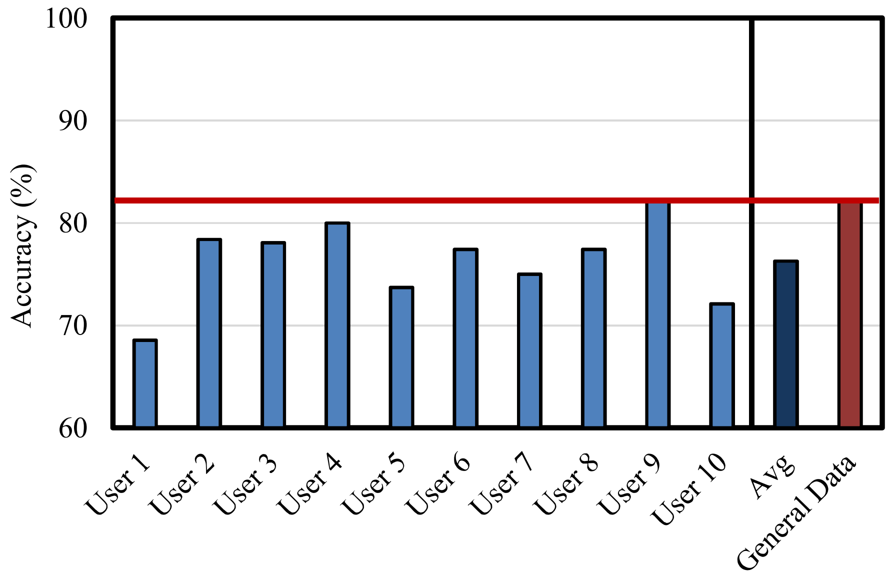


Figure 2.1: Accuracy of the general NIST '19 model on user data.

of 82.1% on the general testing set. However, testing the model against individual user data collected from a smartphone application, the average recognition accuracy drops to 76.1% (Figure 2.1) with less than 70% accuracy for one user. To further illustrate inter-user variation, examples of user data are shown in Figure 2.2, demonstrating how writing styles vary between individual users. This demonstrates the necessity of a model that can quickly adapt to characteristics of a particular user's data.

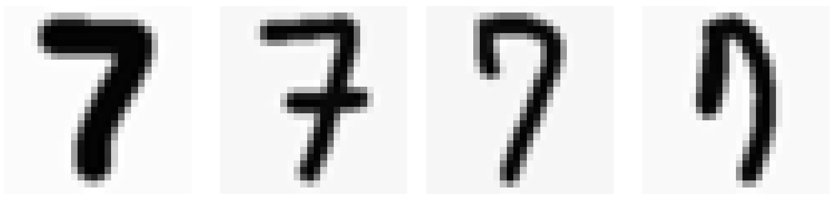


Figure 2.2: The character '7' written by four different users.

Chapter 3

Background

3.1 Deep Neural Networks

Deep learning is a form of machine learning using deep neural networks, here rather than the system being programmed to solve a problem directly, the neural network is trained to learn the relationship between inputs and outputs. This is useful for tasks such as image classification or voice recognition where it is difficult or impossible to decide programmatically which patterns constitute a class or word.

3.1.1 Inference

When using a neural network for classification, the input (e.g. an image) is first input as a list or matrix of values. This input is fed into a layer of neurons, where each input is connected to all neurons and each connection has a trainable weight. The output of a neuron is calculated by first computing the weighted sum of its input connections plus a trainable bias value. A non-linear activation

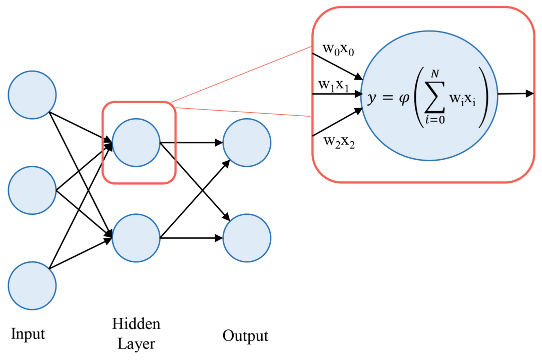


Figure 3.1: Feedforward network diagram

function is then applied to this value, giving the output of the neuron. Non-linear functions are used as they allow the network to represent and compute more complex functions. Several layers of neurons are used, each feeding their output into the next layer. The final layer consists of the N classes that are to be classified, the neuron with the highest output is chosen as the class predicted by the network (see Figure 3.1).

3.1.2 Training

After computing the output of the network, (a class prediction) a loss function is used to measure how incorrect the network is (i.e. the error of the network). This measures how far the network output is from the real output (label). For the classification problem cross entropy loss is the most commonly used loss function, this measures the difference between two distributions. First a softmax function is applied, meaning outputs are scaled so they will sum to 1, this is compared to the label (an array of N values where all classes are zero apart from the correct label which is 1) using cross entropy. This loss value is then used to adjust the weights of the network through backpropogation. Here the gradient of the loss is calculated and fed backwards into the previous layer of neurons which can then use this to calculate its own error gradient. This works backwards through the network until the first layer is reached, hence the name backpropagation [37].

The error gradient determines how much and in which direction the weights should be adjusted. This is also governed by a learning rate, which can also be governed by its own policy. Typically, the learning rate decays after successive training iterations. Initially, weights will need to be adjusted dramatically, however as time goes on, a reduced learning rate allows the weights to be adjusted more delicately to find the precise weights that will result in the lowest loss value.

3.2 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a type of neural network typically used for image processing. Here, instead of a one dimensional input as in traditional feed forward network, 2d and 3d inputs can be used. Rather than having every neuron connected to the next layer of the network, a small window of weights (filter) is moved (convolved) across the input in order to compute the output of a layer. This is known as a convolutional layer. The windows are 3 dimensional and extend over the input channels, however each filter will only generate a 2D output. The outputs of several different filters being convolved over the image are stacked behind one other, resulting in a 3D output (see Figure 3.2). Using convolutional layers preserves the spatial information of the input and has the additional effect of reducing the number of weights required for the network. Typically, earlier layers of the network activate on simple features such as edges and simple shapes whereas later layers activate on features that are composites of these earlier features. Thus allowing the network to build up a complex hierarchical representation of the input images.

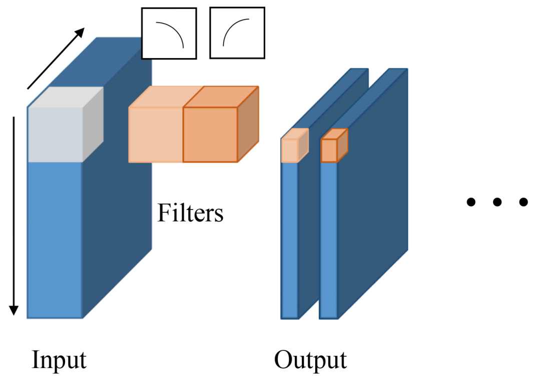


Figure 3.2: Convolutional network diagram

Other layers used in convolutional networks include pooling layers. These, again, move a small window across the input. The output is calculated based on the pooling operation, max or average. When performing max pooling the output value for that window is the largest value in the pooling window, for average pooling the average of the input values is the output value. After pooling the number of channels remains the same, only the width and height of the output is affected. Finally, to compute the final classification of an image a fully connected layer is often used, this flattens out the convolutional output and reduces the dimensions to the number of classes in the classification problems.

3.3 On-Device Acceleration

3.3.1 Hardware Accelerators

DNNs are increasingly used in the embedded domain. However, the computational properties of these are not suited for many existing embedded processors. CPUs are particularly unsuited to DNN acceleration due to the high level of parallelism and lack of control flow in DNNs. Mobile GPUs tend to be a fairly good fit for DNNs device acceleration, however these tend to have a relatively high energy cost. A number of dedicated hardware accelerators have also been proposed, these include DianNao [4], and Eyeriss [6]. These achieve good performance on DNNs but are highly specialized so cannot be used for other forms of processing. Where space is limited and DNN processing is not a primary concern these may not be a particularly good fit. Field Programmable Gate Array (FPGA) processors are also frequently proposed as embedded DNN accelerators [13]. These are not frequently incorporated into embedded devices so again require additional hardware be added. Coarse Grained Reconfigurable Arrays Processors (CGRAs) are an existing class of processor used for multimedia workloads in embedded devices. There has been some work on optimizing these for DNN processing due to their similarity to existing DNN processors [2]. All these accelerators have primarily focused on on-device inference.

3.3.2 Software Optimization

To execute DNNs on embedded devices efficiently, some software based techniques have also been proposed. One of these is Deep Compression [15]. Here DNNs are pruned, quantized and compressed with Huffman coding, in order to reduce their size. Pruning involves removing connections with weights close to zero, the principle behind this is that these connections have little effect on the output of the network so can be safely removed with little impact on accuracy. Quantization is where the precision of each weight is reduced, in Deep Compression this is done through clustering weights and replacing then with indices to the cluster array. This can not be directly optimized to run DNNs on-device but can dramatically reduce the storage size of large networks, making on-device implementations more practical.

There has additionally been work that attempts to compress networks in a manner that can more directly be used hardware accelerators [42]. Along with accelerators that can better exploit these sparse/pruned network structures [33]. FPGA implementations, also, can often exploit the various bitwidth weights generated by quantization.

Additionally, DNN designs such as SqueezeNet [20] and Mobile Nets [19], aim to design DNN structures that are as small as possible while still achieving the same accuracy as larger networks. Techniques such as knowledge distillation [43] can also be used to reduce the size of DNNs, here a smaller network is trained on the outputs of a larger teacher network, rather than on the labels of the dataset itself.

Chapter 4

Methodology

To enable a general DNN model to be adapted to an individual user on-device a DNN network architecture is presented. A large standard network trained on general data is augmented with a small parallel auxiliary network. Together these are able to quickly adapt to user specific data. Here we specifically look at CNNs applied to image classification, although these techniques could be applied to other tasks. A flow diagram of the system is shown in Figure 4.1.

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

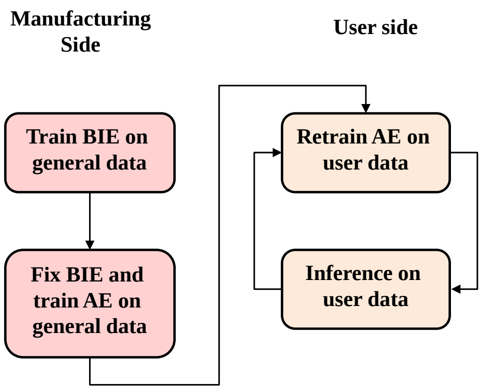


Figure 4.1: Training flow.

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.

4.1 Initialization

Although there has been much work on accelerating on-device inference (i.e. the BIE) there has been little work on developing on-device training accelerators. Therefore, to enable on-device user specialization, the BIE is augmented with a smaller *augmenting engine* (AE) as shown in Figure 4.2. This much smaller network can be executed on an existing general purpose low power processor in parallel. The AE comprises two parts: a simple convolutional network, labeled `C`, followed by a fully-connected layer labeled `B`. The system works as follows: 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 AE is then retrained on the general data to initialize the weights of blocks `B` and `C`.

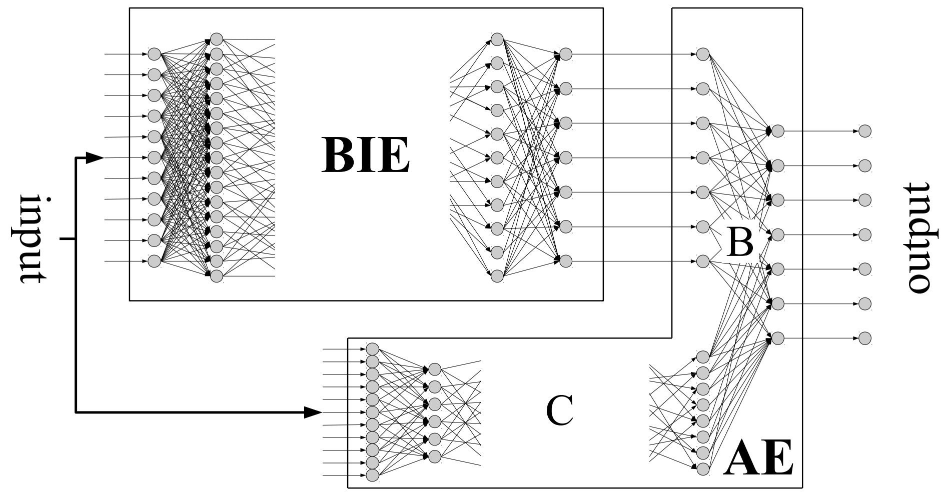


Figure 4.2: The basic inference engine (BIE) with the augmenting engine (AE).

4.2 On-Device Training

The purpose of block `C` is to learn any additional features present in 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 using the image as input for both the BIE and AE blocks, the activations from a hidden layer of the BIE can be used as an input to the AE. This allows the AE to reuse basic features present at earlier layers of the network, while simultaneously reducing the overhead of the AE by reducing the input dimensions.

On the user side, the weights of the BIE remain fixed, and only blocks `B` and `C` are retrained with the user-specific dataset. For the problem of character recognition, through 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 the autocomplete feature used in smartphones. When the user selects a suggested word, this information can be used to label the handwritten character data and transparently perform supervised learning.

Chapter 5

Implementation

The proposed design is applied to two tasks suitable for user customization, Latin and Hangul Handwriting Recognition. For both these problems a publicly available data set and neural network architecture exist.

5.1 Pre-processing

An basic technique that is used to enable user adaptation, is image pre-processing. This aims to make the user and general training data as similar as possible by reducing simple variations. In terms of character recognition, the variations removed by pre-processing are scaling, positioning, and normalization of pixel values. Although this is a simple step it has a large impact on the effectiveness of the system. The techniques used for pre-processing the data are along the lines of the EMNIST dataset [9]. First, characters have Gaussian smoothing applied, are then centered, padded, and scaled down to the relevant input size of the network. For the NIST network this is 28x28 pixels whereas for the Hangul network this is 64x64 pixels. EMNIST is not directly used for training the NIST network as we wished to ensure that the exact same pre-processing procedure was applied to both the general and the user-specific datasets.

5.2 Latin Handwritten Character Recognition

5.2.1 Dataset and BIE Selection

For Latin character classification, the NIST '19 dataset [12] is chosen. This consists 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 implementation, the well known LeNet-5 [28] is used, modified to produce 62 outputs, rather than the standard ten. Alternative implementations for this base network exist, e.g. the network described in [8]; however, this is based on a committee of multiple networks working together in parallel, rendering it less suitable for the embedded domain.

5.2.2 AE Design

When applying the BIE-AE design a number of factors were considered. The starting AE design is chosen as a basic convolutional network consisting of one convolutional layer, one pooling layer and one fully connected layer. When evaluating the AE design a two design variables are investigated, these being the number of channels in the convolutional layer, the second being average pooling to downsample the input. Configurations with one, five, 10, 20, and 50 channels were explored. When investigating downsampling three settings are tested: no downsampling, downsampling by half, downsampling to a quarter of the original input size. Table 5.1 shows how the accuracy and overhead of the AE designs change in relation to these two parameters. Figure 5.1 visualizes

Table 5.1: Overhead and accuracy for different Latin AE designs.

The selected configuration is shown in a **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	1,582	74.5	92.9
	50	1,622	3,904	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

this data. Two overheads are import to assess when choosing a design, the first of these is the number of MAC operations, this is proportional to the execution time of the network. The second is the memory, this determines whether the network design fits in the on chip memory of the general purpose accelerator, if the memory required is too large a number of off chip memory accesses will be required. These are costly in terms of energy efficiency, so should be minimal to ensure a low power AE design. The *MAC* and *Memory* columns give the total memory requirements of the weights and activations used when training. To determine the influence of these design variables on the accuracy of the model, *start accuracy* and *end accuracy* are listed. These show the accuracy of the entire network before and after user specialization.

Table 5.2: Absolute and relative overhead of the Latin AE with respect to the BIE.

	Training			**Inference**		
	BIE	**AE**	**Ratio**	**BIE**	**AE**	**Ratio**
MAC (kOps)	4,376	64	2%	2,319	44	2%
Activations (kB)	155	15	10%	79	9	12%
Weights (kB)	3,652	157	4%	1,826	78	4%

The table and the figure show that downsampling (average pooling) has the largest 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. However, 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. Figure 5.2 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 5.2 gives the absolute and relative overhead of the AE to the BIE on a single image using single-precision floating point numbers for weights and activations. Training and inference overheads are minimal when compared to the relatively simple BIE.

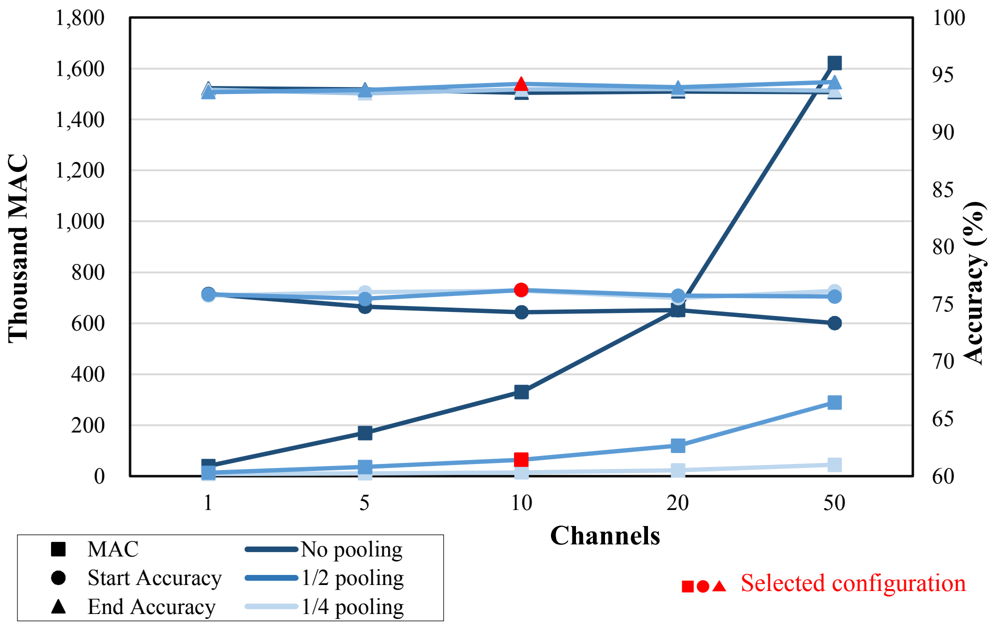


Figure 5.1: Overhead and accuracy for Latin AE designs with BIE activated.

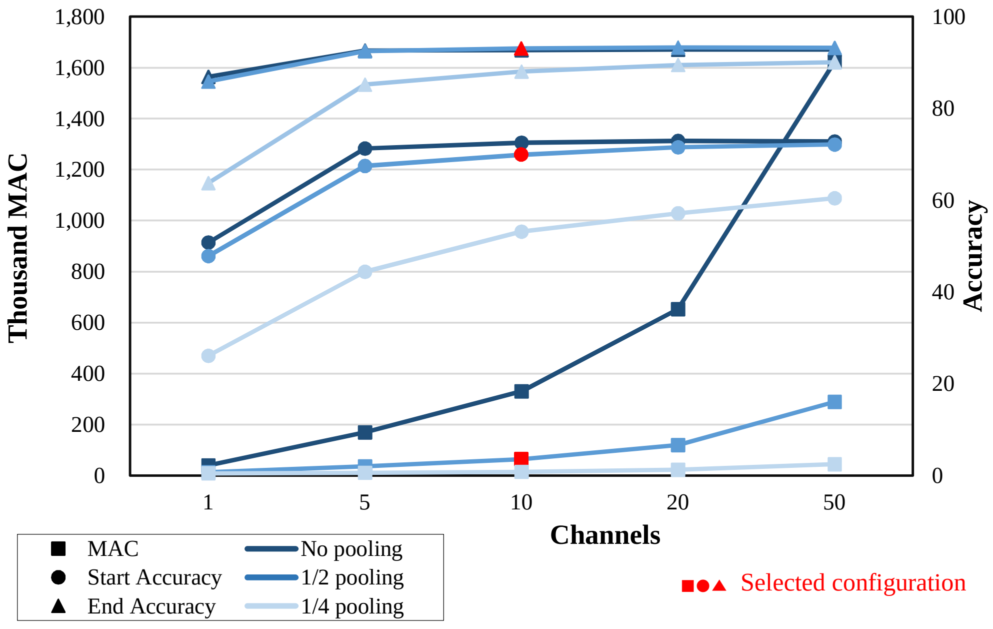


Figure 5.2: Overhead and accuracy for Latin AE standalone designs.

Figure 5.3 (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 convolutional 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.

5.3 Korean Handwritten Character Recognition

5.3.1 Dataset and BIE Selection

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 [21] is used. This consists of the 520 most commonly used handwritten Korean characters, with approximately 1000 examples for each character. The network described in [25] 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 [34]. This, if implemented as a dedicated hardware chip, would make a good component in the BIE-AE architecture.

5.3.2 AE Design

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 required to train the AE. This is due to the large number of weights required for a fully

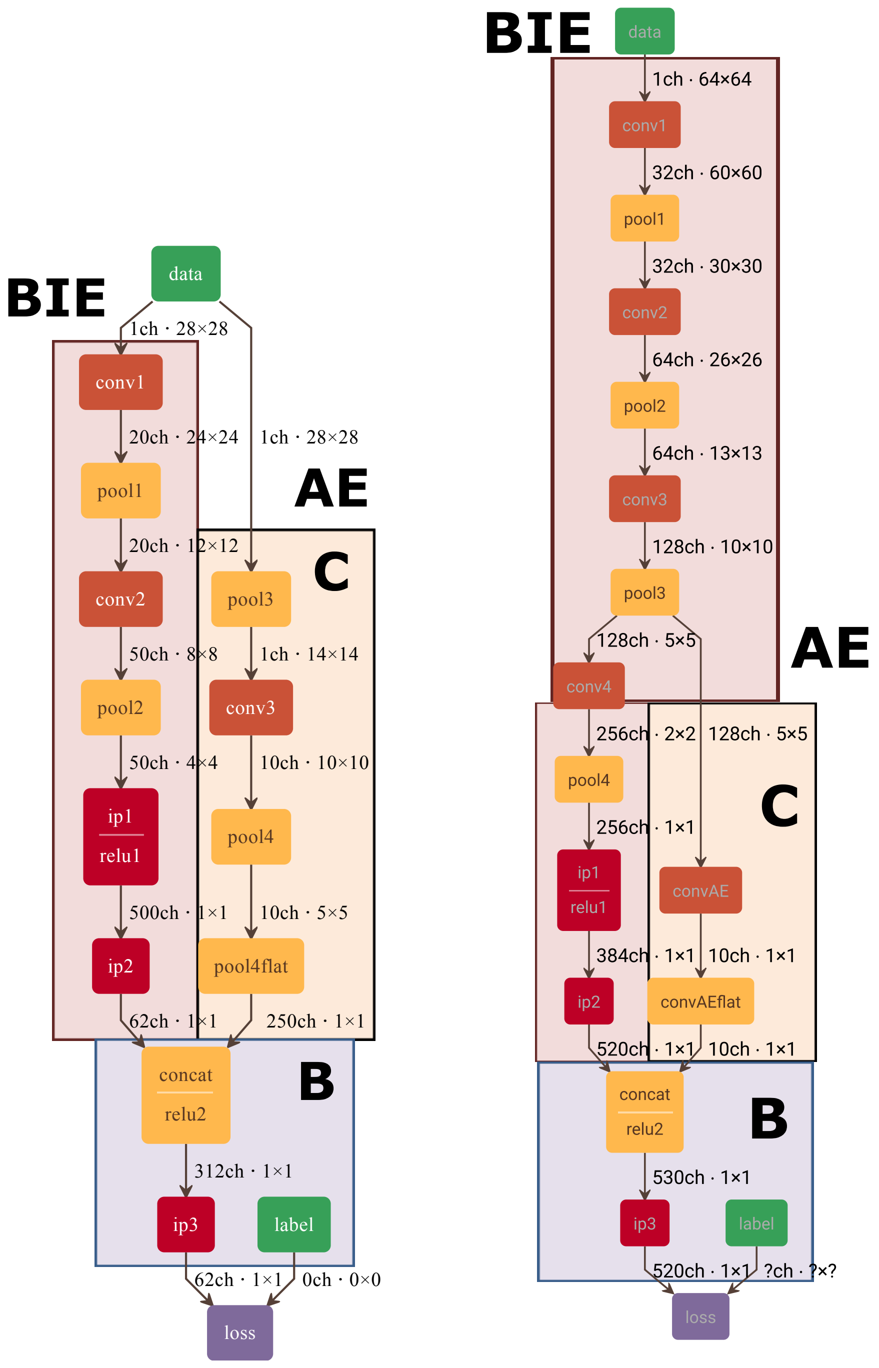


Figure 5.3: Latin and Hangul BIE-AE Implementations.

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 here will add 520 weights. Therefore, using an input from an intermediate layer of the BIE is investigated. This allows the AE to use much smaller inputs, resulting in much smaller outputs, along with the additional advantage of being able to reuse the features earlier in the BIE. Therefore, the two variables investigated for this design are the layer of the BIE that serves as the input to the AE and the number of channels used in the convolutional layer. In Figure 5.3 (b), the locations considered as inputs to the AE are the original image and after `pool1`, `pool2`, and `pool3` in the BIE. The fourth layer is not considered because its output dimension of 1×1 pixel does not allow for further convolutions, making it difficult to learn new spatial features in the user data. The considered designs are shown in Table 5.4.

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 that results from flattening a large convolutional output. This large number of weights results in a model that is both difficult to train and has a high training overhead. The design using the input raw image with 50 convolutional channels was unable to be trained even on a standard GPU due to its high memory demands. For the 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 Figure 5.3 (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 5.3 lists the absolute and relative overhead of

Table 5.3: 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)	1,746	17	1.0%	881	15	1.7%
Weights (kB)	8,042	2,461	31.0%	4,021	1,230	31.0%

the AE relative to the BIE 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.

Table 5.4: Hangul AE designs

Input location	**Input dimensions**	**Channels**	**MAC (kOps)**	**Memory (kB)**	**Start Accuracy (%)**	**Final Accuracy (%)**
Image	64x64x1	1	4,375	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	30x30x32	1	1,785	5,107	90.8	98.0
		5	6,764	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	13x13x64	1	756	2,561	90.5	97.6
		5	1,618	3,963	91.0	97.1
		10	2,695	5,715	89.3	97.5
		20	4,850	9,219	87.0	97.1
		50	11,313	19,731	86.3	95.8
Pool 3	**5x5x128**	1	548	2,210	93.3	98.4
		5	578	2,329	91.2	98.5
		10	**615**	**2,478**	**92.0**	**97.9**
		20	690	2,776	92.1	98.5
		50	913	3,669	92.5	98.6

Chapter 6

On-Device Acceleration

The BIE-AE structure is designed to take advantage of existing low-power general-purpose mobile accelerators for training, while using a higher power implementation for inference, this being either a high-performance general-purpose accelerator or dedicated hardware. To demonstrate the effectiveness of using a low power general purpose embedded accelerator to execute the AE an existing accelerator is used to evaluate the network. The processor chosen is the Samsung Reconfigurable Processor (SRP), due to its incorporation into several commercial devices. However, other existing embedded accelerators such as FPGAs could also be used. This is a Coarse Grained Reconfigurable Array (CGRA) processor, used primarily to accelerate multimedia workloads in embedded devices [35, 24, 30, 29]. To accelerate the BIE the NVIDIA Jetson TX2 [32] is chosen as a representative accelerator, this is a high powered state of the art development board for DNN processing using a mobile GPU.

The SRP consists of an array of heterogeneous processing elements (PEs) and register files (RFs), connected with an interconnected data-flow network.

Fast on-chip data storage is provided by SRAM or eDRAM, a schematic diagram is shown in Figure 6.1 (a). The SRP is chosen to accelerate the AE as it has many similarities to recently proposed dedicated DNN accelerators. First, in both, a large on chip memory is situated close to the PEs to allow for the fast loading of DNN weights/multimedia workloads. Second, a dataflow network is used to move data between PEs. Third, multiple PEs are used to process input in parallel. If the weights and activations of the AE fit on the on-board SRP memory, expensive off-chip memory accesses can be avoided.

6.1 Architecure Optimizations

In its basic configuration, the SRP is fairly suited to process DNNs, however as this is a design time reconfigurable processor a number of configurations more suited to DNN processing are also explored. The basic configuration is a 32-bit floating-point CGRA with 4×4 heterogeneous PEs and a total of 320KB 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.

To ensure the SRP can continue to be used as a general-purpose accelerator and to avoid increasing power usage and chip size, modifications to the architecture should be minimal. Therefore, adding PEs is not considered. Primarily, the number and positioning of floating point and memory units along with the amount of on-chip memory is the architectural design space explored. This is due to the specific properties of CNN layers which consists primarily of matrix multiplications of floating point values, meaning that floating point and memory operations dominate the overhead.

To accommodate the frequency of these instructions, the following hardware modifications to the SRP are explored. The on-chip SRAM is replaced by 4MB

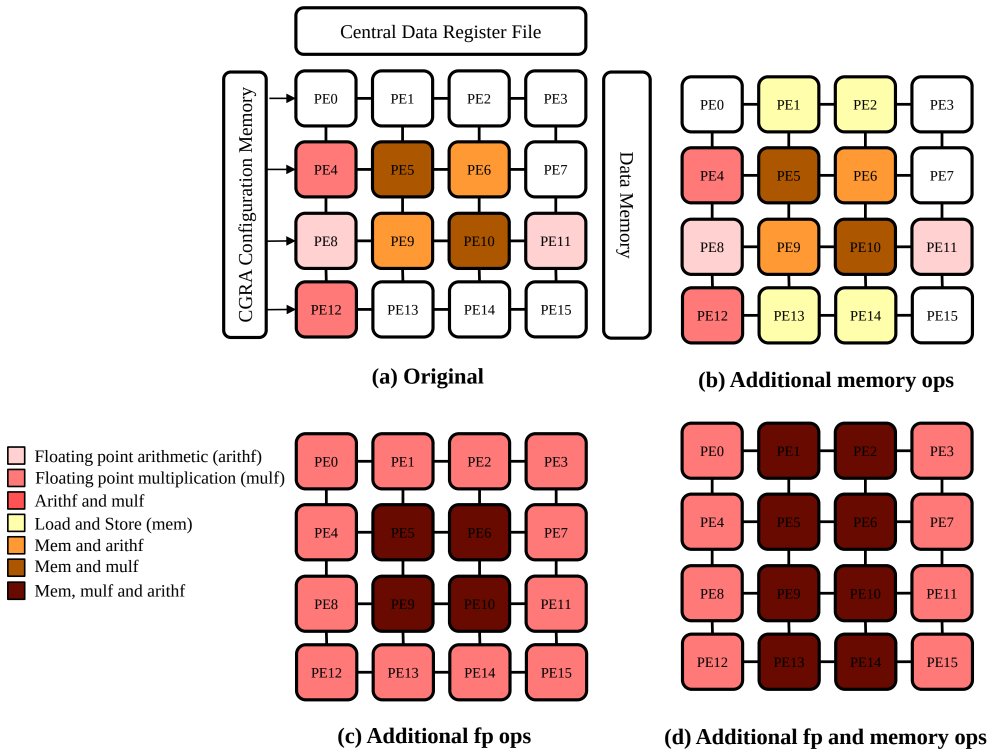


Figure 6.1: SRP hardware configuration variations.

of 3D-stacked eDRAM [31] comprising of eight banks in a similar layout as proposed by DianNao [4] 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 Figure 6.1. These involve increasing the number of memory-enabled PEs, the number of floating point-enabled PEs, or both.

6.2 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 deep learning frameworks like Caffe [22] or Tensorflow [40] depend heavily on libraries. For this reason, an DNN framework supporting embedded accelerators such as CGRAs was used [23]. This is written in the C language, without any external dependencies. Compiler optimizations are also explored, for this the auto-tuning framework proposed in [2] is used. This automatically performs architecture aware loop unrolling, fusion, and interchange as well as using a CGRA optimized convolution algorithm. The framework requires the the C code description of the DNN and the CGRA architecture description to perform these optimizations.

Chapter 7

Experimental Setup

The proposed BIE-AE networks are trained first with a publicly available general dataset. An Android application was developed and used to collect data from individual users, in order to test user customization. For Latin handwriting recognition, 40 images of each of the 62 character classes were collected from 10 users. Of these 40 image sets, 30 are used to train the AE and the remaining 10 sets are used as a testing set, giving 1860 training images and 620 testing images per user. For Hangul recognition, due to the large alphabet size a subset of 50 of the most commonly used characters were collected from seven users. Here, 20 sets are collected with 10 being used for training and the other half comprising the test set.

The BIE and AE are first trained with the relevant general purpose dataset, the BIE is then fixed and the AE is retrained with the user-specific dataset one set at a time. To implement the network design for Latin character classification the Caffe framework is used, the user data is pre-processed using a Matlab script and packaged using a bash script. To implement Hangul classification

Table 7.1: Overview of compared processor architectures.

Architecture	**Type**	**PEs**	**Clock (MHz)**	**Power (mW)**	**Technology (nm)**
ARMv8-A [1]	general-purpose CPU	4	1200	271	20
VLIW [38]	general-purpose accelerator	3	500	50	32
CGRA	general-purpose accelerator	16	500	150	32
Jetson TX2 [32][1]	mobile GPU	256	850	4,800	20

the Tensorflow framework was used, again a Matlab script was used to pre-process the images which were again packaged using a bash script.

To test the efficiency of the network on an embedded device, the structure is rebuilt in C using the CGOOD framework [23]. This is evaluated on the various embedded hardware processors show in Table 7.1. The accelerators chosen are the ARMv8, a general purpose embedded CPU running inside a Raspberry Pi 3, this is chosen due to its wide use in mobile devices. Second, the SRP chip running in VLIW mode is used, this is an extremely low power configuration so gives some insight into very power restricted embedded processors. Third, the SRP with the CGRA enabled is used, this shows the performance of a low power chip, but one that is somewhat suited to DNN processing. Finally, to evaluate high power state of the art embedded DNN processors the Jetson TX2 is used, this is a mobile GPU based development board specifically for DNN processing. This gives a fairly large range of different types of embedded processors that may be used to process DNNs.

The latency results are determined by training or inferencing on one image using the CGOOD implementation of the network. Measurements for the ARMv8 and Jetson have been executed on real systems; for the CGRA and

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

VLIW results, a cycle-accurate simulator is used that simulates both the core and the memory. Energy is computed by multiplying the average power consumption from Table 7.1 with the runtime of the relevant architecture.

Chapter 8

Evaluation

This section evaluates the performance of the presented BIE-AE network structure with respect to user specialization, as well as evaluating its suitability for execution on existing embedded systems.

8.1 Latin Handwritten Character Recognition

Figure 8.1 shows the gain in accuracy for individual users as the AE is retrained on successive user datasets. Each training set consists of one image from each class and is trained on for 10 epochs. After 30 training sets, the average accuracy improves from 76.1% to 93.7%. Table 8.1 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.

In this problem domain, errors predominantly stem from similar looking

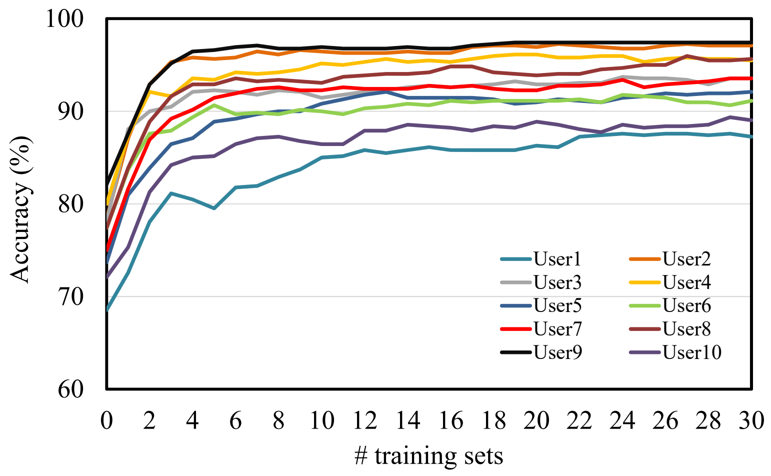


Figure 8.1: Accuracy of BIE + AE on Latin user data

characters. These include characters where the lower and upper class symbols are similar (e.g. f, F) and similarly shaped characters (i,l,1,j). For an individual user, there is usually some consistency in the way these characters are written, so a personalized model is able to perform better. However, a general model must be able to distinguish the characters from several different users so can't exploit the internal consistency of a single user's writing style. These improvements accounts for the large improvements in the 'all' and 'letters' categories where many of the misclassified characters are of this type.

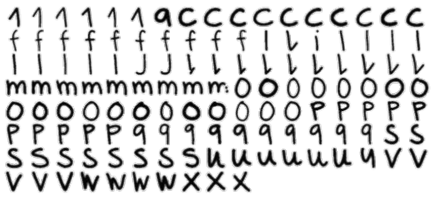
To illustrate how the BIE-AE structure adapts to a user's writing style, Figure 8.2 shows the misclassified characters after training with 0, 1, 2, 3, 5, and 10 sets of user data. We observe that the way the network improves is somewhat systematic. Figure 8.2 (a) shows all characters that the general model failed to correctly classify. After one training set (b), the misclassified m's all become correctly classified as well as many of the x, c, u, w, v, p, and 1 characters, giving an overall improvement in recognition, which can be seen as a general adaptation to the user's style. For subsequent steps, improvements are

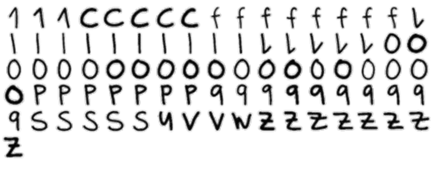
Table 8.1: Accuracy before/after retraining the Latin BIE-AE model using individiual user data.

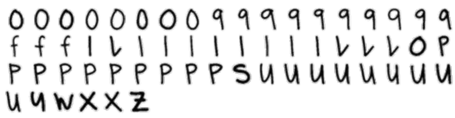
	all		**letter**		**lower**		**upper**		**digit**	
Dataset	**before**	**after**	**before**	**after**	**before**	**after**	**before**	**after**	**before**	**after**
NIST	82.1	73.9	69.7	73.8	88.8	87.0	96.2	96.1	98.2	96.7
User 1	67.7	87.6	74.6	91.2	86.5	96.2	95.8	98.5	97.0	99.0
User 2	76.9	97.6	78.1	98.7	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**

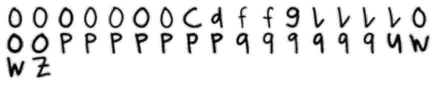
more concentrated in certain areas. After the second training set (c), the system learns how to distinguish between upper and lower case o's, however, as a result there is an increase in 0's being misclassified. A similar pattern can be observed between q's and 9's. There is often an oscillation between similar characters, where after one set one class is over predicted and after the following another is over predicted, but as more data is collected these similar characters are distinguished more accurately. These difficult to distinguish characters eventually become more accurately classified, as can be seen in Figure 8.2 (f). After 10 training sets 0,o and Os along with 9 and q are better classified and the balance of misclassification is more evenly spread between the classes.

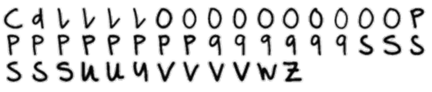
The percentage of images misclassified by the retrained models but correctly


(a) 0 sets


(b) 1 set


(c) 2 sets


(d) 3 sets


(e) 5 sets

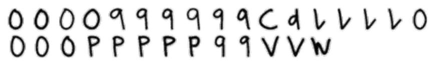

(f) 10 sets

Figure 8.2: Misclassified images from Latin test set while retraining

classified by the general model is shown in Figure 8.3. This demonstrates the errors introduced by retraining the system. Initially, the images that are incorrectly classified by the system are very similar to the ones misclassified by the general model. Quickly, the misclassified images diverge from the initially misclassified images. However, the percentage of newly introduced misclassifications always accounts for less than 50% of the total incorrect images, meaning that the majority of misclassified images are difficult for both the general and personal models to classify correctly.

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 user data. 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. More

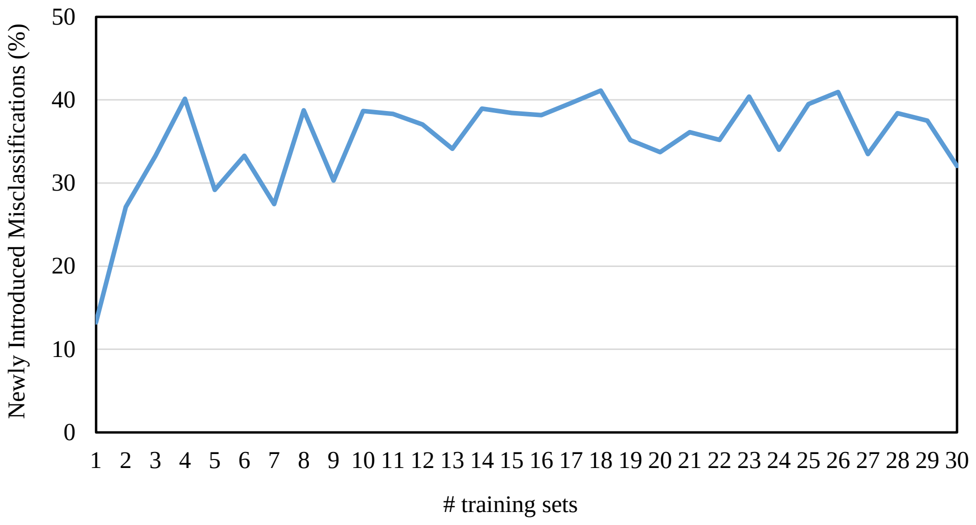


Figure 8.3: Percentage of misclassified images introduced by retraining

training samples are also required for the accuracy to reach a satisfactory level, which is particularly relevant for the user customization scenario. For end users it is important to have a high starting accuracy 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.

There exist also some limitations with regards to personalization, after specializing on user data the accuracy on the general dataset falls. In this case, from 82.1% on average to 73.9%. This is unsurprising due to the nature of personalization, when a model becomes specialized to a user it can no longer classify general data as accurately. To overcome this problem the initial weights of the AE can be saved and reloaded if the user of the device changes. Due to the small size of the AE the overhead of storing this is minimal.

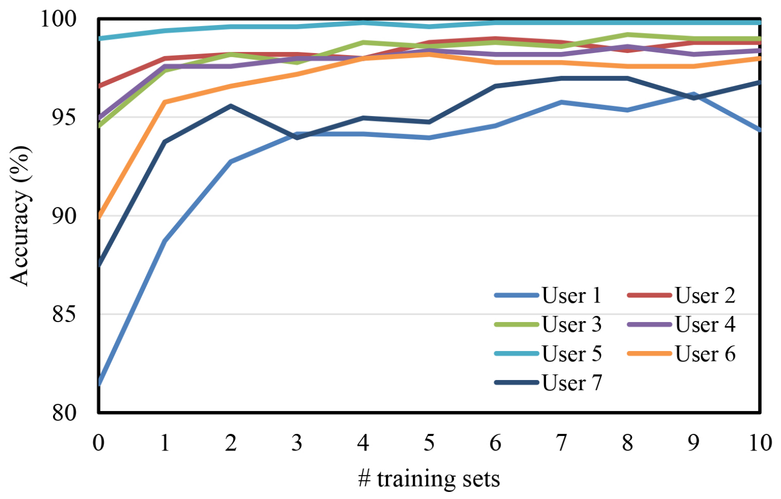


Figure 8.4: Accuracy of BIE + AE on Hangul user data

8.2 Korean Handwritten Character Recognition

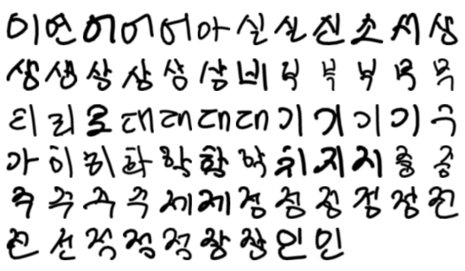
Figure 8.4 shows how the accuracy for Hangul characters increases for individual users with an increasing amount of user data. Each training set consists of one character from each collected class and is trained on for 100 epochs to increased the speed of learning. The average starting accuracy on the user data is considerably higher than for Latin characters at 92.0%, but still improves significantly to 97.9% as additional user data is added. The higher starting accuracy is likely due to the reduced number of hard to distinguish classes in Hangul compared to Latin. As the characters of Hangul are more complex than the Latin alphabet the number of properties that can be used to distinguish them are more numerous. Harder to distinguish categories in this domain typically stem from characters where the majority of component characters are the same but one differs e.g. [정, 장, 작]. The detailed accuracy for each individual user is listed in Table 8.2.

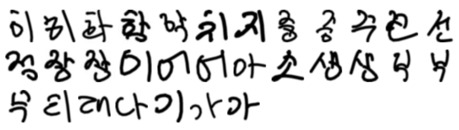
Table 8.2: Accuracy before/after retraining the Korean BIE-AE model using user-specific data.

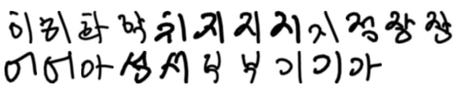
Dataset	**before**	**after**
SERI95a	95.3	90.8
User 1	81.5	94.4
User 2	96.6	98.8
User 3	94.6	99.0
User 4	95.0	98.4
User 5	99.0	99.8
User 6	89.9	98.0
User 7	87.5	96.8
Average	92.0	**97.9**

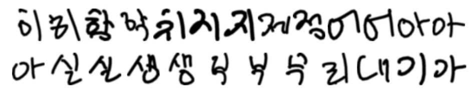
Here we see that the accuracy for the personalized models on the general dataset is again reduced after specialization, this time by 4.5% compared to the 8.2% reduction for the Latin problem. This is most likely due to the difference in initial accuracy for the users. The Latin users have a lower starting accuracy, meaning that the model needs to be adjusted more in order to classify their data correctly. Whereas the higher starting accuracy of the Hangul model means that the general model is adjusted less, meaning it is still able to classify the initial dataset correctly. This is confirmed by looking at individual user data. The general test accuracy for user 1 and 7, the two users with the lowest start accuracy, is around 87% after specialization, whereas for all other users it is above 90%.

Figure 8.5 shows the test characters that are misclassified by the Hangul network for one user after successive training sets. This shows that initially that characters such as [아, 어, 이] are often confused by the general model. This is also the case for the characters [래, 대], [생, 상], [학, 항, 함, 막] and [정, 적, 점,


(a) 0 sets


(b) 1 set


(c) 2 sets


(d) 5 sets

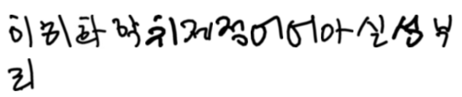

(e) 10 sets

Figure 8.5: Misclassified images from Hangul test set while retraining

잔, 항]. A lot of these confusions are eliminated after the first retraining set as the network gets better at telling apart these hard to distinguish characters. Similar to the Latin example, the greatest gain in accuracy is this first set and this improvement is fairly evenly distributed across classes. Later training sets have less effect on the accuracy as the scope for improvement is reduced by the initial training set, where the accuracy already increases from 87.5% to 93.8%.

8.3 On-Device Acceleration

To determine the ideal hardware configuration for the modified SRP processor, the design variations (Figure 6.1) are evaluated using the BIE design for the Latin network. This is chosen rather than the AE as larger CNNs should show optimizations more clearly. The performance in terms of speedup compared to

Table 8.3: Comparison of different hardware configurations.

Figure 6.1	**(a) original**	**(b) add mem**	**(c) add fp**	**(d) add mem & fp**
Speedup	33.9	46.1	37.6	46.7

executing the BIE on the VLIW portion of the hybrid processor is shown in Table 8.3. The results show that the addition of more memory units yields the greatest increase in performance, while additional floating point units improve performance only marginally. Design (b) is used for further experiments as, although it is slightly slower than design (d), it is closer to the original SRP and less specialized, thus better suited for general-purpose acceleration.

The modified SRP is compared to an ARMv8 processor, a 3-issue VLIW, and the Jetson TX2 mobile GPU. The processing time of these various processors is shown in Table 8.4, and as a graph in Figure 8.6. This shows that the chosen CGRA used for the AE implementation is sufficiently fast 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 (Figure 5.3), the latency of a single operation may be slightly higher, however, for batch processing the AE should not slow down the BIE. For Hangul inference, for example, the BIE on the Jetson TX2 takes $1ms$ to execute, but the results of the AE on the CGRA are available after $0.7ms$.

We also observe that the 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. The reason for this is that the Jetson is specialized for both inference and comparatively large networks, so retraining a

Table 8.4: 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

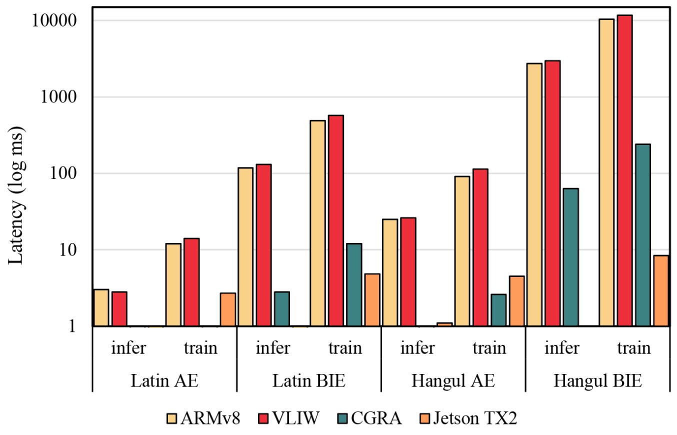


Figure 8.6: Processing time of BIE and AE on different architectures

small network is not necessarily something it is well suited to. This shows that using a low power processor such as the CGRA makes sense for this component.

A comparison of the energy consumption for inference and training on one image is given in Table 8.5. For the chosen CGRA, energy is reduced 47 times compared to the ARM processor for the Latin AE. 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

Table 8.5: 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

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.

Chapter 9

Related Work

The proposed methodology is an example of the domain adaptation problem, which is related to transfer learning. It makes use of transfer learning techniques but uses an architecture that can effectively exploit existing embedded hardware accelerators.

Several other approaches to domain adaption have been proposed, however this one targets embedded environments specifically. Other approaches to domain adaption include Adversarial Discriminative Domain Adaption (ADDA)[41], this is an unsupervised approach that uses two separate networks for both the source and target domains, (general and user data) and a discriminator 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 learns to map the new target images to the source feature space, while the discriminator attempts to distinguish the (feature) 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. Essentially, the target network attempts to mimic the source network given a target input. However, this process requires both general and user data to be available when training which is not possible for the proposed use case. Additionally, the target network is typically a duplicate of the source network, giving it a large network structure rendering it unsuitable for the embedded domain.

Alternative methods of adapting DNNs to a new task are the transfer learning techniques of Joint Training [3] and Fine-Tuning [10]. In joint training, the entire CNN is retrained with both general and user datasets to allow it to be used 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 one advantage of the proposed structure. Additionally, the parallelism of the BIE allows the AE to be trained on a lower powered accelerator giving a more efficient training implementation.

Federated learning is also a technique that bears some resemblance to the presented work [27]. This works by training on a distributed network of user devices, each user device having a subset of user data (and classes). The on-device model is trained on this data. The weights from all of these models are sent to a server, where they are averaged, the averaged model is then distributed again to all the distributed devices. This happens for a number of rounds resulting in a general model that is able to recognize data from all distributed devices. Federated learning does not, however, make use of the properties of personalization and instead creates a better general model through distributed learning.

There have additionally 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 [20] and the base network used to recognize Hangul in this paper [34]. The proposed focus on the CGRA is to make use of a low power accelerator already present in mobile devices as a general purpose accelerator, primarily for the training task. There has been work on accelerating DNNs with CGRAs [39] however, this proposes significant hardware changes to the CGRA, rendering it suitable only for CNN acceleration. What are proposed here are minor modifications to the CGRA, allowing it to be used as both an effective DNN and general purpose accelerator on embedded devices.

Chapter 10

Conclusion

In conclusion, this work proposes a DNN architecture suitable for on-device personalization of deep neural networks. This consists of a large base inference engine (BIE) trained on general data and a smaller augmenting engine (AE) that can be retrained on the device. This allows for an energy efficient method of adapting to additional user data on-device.

The proposed approach is applied to the problem of handwritten character recognition for both the Latin and the Korean alphabet. After user specialization, the system has a 3.5-fold lower classification error than the standalone BIE, increasing classification accuracy from 76.1 to 93.7% for the Latin alphabet and from 92.0 to 97.9% for the Korean character set. Experiments on various embedded accelerators show that when using a high powered inference accelerator for the BIE and a low-power general purpose accelerator for the AE efficient training is achieved, with a minimal effect on the inference overhead.

Bibliography

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

[2] I. Bae, B. Harris, H. Min, and B. Egger. Auto-tuning CNNs for coarse-grained reconfigurable array-based accelerators. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(10):1–1, Oct 2018.

[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] 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.

[6] 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.

[7] D. Ciregan, U. Meier, and J. Schmidhuber. Multi-column deep neural networks for image classification. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3642–3649, June 2012.

[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] 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.

[11] A. Graves, A. r. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6645–6649, May 2013.

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

[13] D. Gschwend. Zynqnet: An fpga-accelerated embedded convolutional neural network. Master's thesis, Dept. of Information Technology and Electrical Engineering, Swiss Federal Institute of Technology, Zurich (ETH Zurich), 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] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[16] B. Harris, I. Bae, and B. Egger. Architectures and algorithms for on-device user customization of cnns. *Integration*, 2018.

[17] B. Harris, M. S. Moghaddam, D. Kang, I. Bae, E. Kim, H. Min, H. Cho, S. Kim, B. Egger, S. Ha, and K. Choi. Architectures and algorithms for user customization of cnns. In *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 540–547, Jan 2018.

[18] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, Nov 2012.

[19] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[20] 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.

[21] K. Injung. Hanguldb (seri95a, pe92). `https://github.com/callee2006/HangulDB`, 2018.

[22] 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.

[23] D. Kang, E. Kim, I. Bae, B. Egger, and 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.

[24] C. Kim, M. Chung, Y. Cho, M. Konijnenburg, S. Ryu, and J. Kim. ULP-SRP: Ultra low power samsung reconfigurable processor for biomedical applications. In *International Conference on Field-Programmable Technology (FPT)*, pages 329–334. IEEE, 2012.

[25] I.-J. Kim and X. Xie. Handwritten hangul recognition using deep convolutional neural networks. *International Journal on Document Analysis and Recognition (IJDAR)*, 18:1–13, 2014.

[26] I.-J. Kim and X. Xie. Handwritten hangul recognition using deep convolutional neural networks. *International Journal on Document Analysis and Recognition (IJDAR)*, 18(1):1–13, Mar 2015.

[27] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.

[28] 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.

[29] J. Lee, Y. Shin, W.-J. Lee, S. Ryu, and K. Jeongwook. Real-time ray tracing on coarse-grained reconfigurable processor. In *International Conference on Field-Programmable Technology (FPT)*, pages 192–197, Dec 2013.

[30] W.-J. Lee, S.-H. Lee, J.-H. Nah, J.-W. Kim, Y. Shin, J. Lee, and S.-Y. Jung. SGRT: a scalable mobile gpu architecture based on ray tracing. In *ACM SIGGRAPH 2012 Posters*, page 44. ACM, 2012.

[31] R. E. Matick and S. E. Schuster. Logic-based edram: Origins and rationale for use. *IBM Journal of Research and Development*, 49(1):145–165, Jan 2005.

[32] NVIDIA Corporation. Nvidia jetson tx2 delivers twice the intelligence to the edge˙ `https://devblogs.nvidia.com/jetson-tx2-delivers-twice-intelligence-edge/`, Sep 2018.

[33] A. Page, A. Jafari, C. Shea, and T. Mohsenin. Sparcnet: A hardware accelerator for efficient deployment of sparse convolutional networks. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 13(3):31, 2017.

[34] 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.

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

[36] A. S. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson. CNN features off-the-shelf: an astounding baseline for recognition. *CoRR*, abs/1403.6382, 2014.

[37] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1. chapter Learning Internal Representations by Error Propagation, pages 318–362. MIT Press, Cambridge, MA, USA, 1986.

[38] 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.

[39] 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.

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

[41] E. Tzeng, J. Hoffman, K. Saenko, and T. Darrell. Adversarial discriminative domain adaptation. *CoRR*, abs/1702.05464, 2017.

[42] M. Wess, S. M. P. Dinakarrao, and A. Jantsch. Weighted quantization-regularization in dnns for weight memory minimization toward hw implementation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(11):2929–2939, 2018.

[43] J. Yim, D. Joo, J. Bae, and J. Kim. A gift from knowledge distillation: Fast optimization, network minimization and transfer learning. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, 2017.

요약

내장형 기기에서 심층 신경망을 추론할 수 있는 아키텍처들은 존재하지만 내장형 기기에서 신경망을 학습하는 연구는 별로 이뤄지지 않았다. 실제 환경을 반영하는 학습용 데이터 집합을 모으는 것이 어렵고 사용자간의 다양성으로 인해 일반적으로 학습된 모델이 충분한 정확도를 가지기엔 한계가 존재하기 때문에 사용자 맞춤형 심층 신경망이 필요하다. 이 논문에서는 기기상에서 저전력으로 사용자 맞춤화가 가능한 심층 신경망 아키텍처를 제안한다. 이러한 접근 방법은 라틴어와 한글의 필기체 글자 인식에 적용된다. 라틴어와 한글에 사용자 맞춤화를 적용하여 일반적인 데이터로 학습한 심층 신경망보다 3.5배나 작은 예측 오류의 결과를 얻었다. 또한 이 아키텍처의 실용성을 보여주기 위하여 다양한 내장형 프로세서에서 실험을 진행하였다.

주요어: Deep Learning, On-Device Learning, User Adaptation

학번: 2016-25821

Acknowledgements

I would first like to thank our professor Bernhard Egger for his guidance and support in performing this research. I am thankful for the many opportunities provided to publish work in both conferences and journals in my time at the Computer Systems and Platforms Lab.

Second, I would like to thank Inpyo for his help and support on the many papers we worked on together as well as with the Samsung project that we both worked on. Although it was often a struggle, it was thanks in no small part to his hard work that we were able to finish work with very tight deadlines.

I would also like to thank all the other lab members for making my time at the lab enjoyable. I'd like to thank 영민, 영현, 창연, 영수, 지웅, 혜미, 민수, Simeon, and Richard. Additionally, I'd like to say thank you to the lab members who were at the lab at the same time as me but have already graduated, so thank you to 수림, 창민, 호찬, Camilo, An na, and Xinyi.

I would also like to thank NIIED for funding my masters program through the Korean Government Scholarship Program. Without this I would not have been able to complete my degree.

Finally, I would like to thank my girlfriend and my family for their support during my time in Korea and at the lab.