Big/Little Deep Neural Network for Ultra Low Power Inference

Eunhyeok Park[1]
canusglow@gmail.com

Dongyoung Kim[1]
dongyoung42@gmail.com

Soobeom Kim[1]
Soobeom15@gmail.com

Yong-Deok Kim[2]
yd.mlg.kim@samsung.com

Gunhee Kim[3]
gunhee@snu.ac.kr

Sungroh Yoon[4]
sryoon@snu.ac.kr

Sungjoo Yoo[1]
sungjoo.yoo@gmail.com

Computing Memory Architecture Lab, Seoul National University[1]
Software R&D Center, Device Solutions, Samsung Electronics[2]
Vision and Learning Lab, Seoul National University[3]
Advanced Computing Lab, Seoul National University[4]

ABSTRACT

Deep neural networks (DNNs) have recently proved their effectiveness in complex data analyses such as object/speech recognition. As their applications are being expanded to mobile devices, their energy efficiencies are becoming critical. In this paper, we propose a novel concept called big/LITTLE DNN (BL-DNN) which significantly reduces energy consumption required for DNN execution at a negligible loss of inference accuracy. The BL-DNN consists of a little DNN (consuming low energy) and a full-fledged big DNN. In order to reduce energy consumption, the BL-DNN aims at avoiding the execution of the big DNN whenever possible. The key idea for this goal is to execute the little DNN first for inference (without big DNN execution) and simply use its result as the final inference result as long as the result is estimated to be accurate. On the other hand, if the result from the little DNN is not considered to be accurate, the big DNN is executed to give the final inference result. This approach reduces the total energy consumption by obtaining the inference result only with the little, energy-efficient DNN in most cases, while maintaining the similar level of inference accuracy through selectively utilizing the big DNN execution. We present design-time and runtime methods to control the execution of big DNN under a trade-off between energy consumption and inference accuracy. Experiments with state-of-the-art DNNs for *ImageNet* and *MNIST* show that our proposed BL-DNN can offer up to 53.7% (*ImageNet*) and 94.1% (*MNIST*) reductions in energy consumption at a loss of 0.90% (*ImageNet*) and 0.12% (*MNIST*) in inference accuracy, respectively.

Categories and Subject Descriptors

C.1.3 [Other Architecture Styles]: Neural nets.

General Terms

Management, Performance, and Design.

Keywords

Deep neural network, low power.

1. INTRODUCTION

Deep neural networks (DNNs) are becoming more and more popular since they can address large-scale problems better than existing solutions [1,2,4,6]. Commercial server-level DNN applications are being actively developed, e.g., [5]. Moreover, DNNs are also expected to be applied to mobile devices such as smartphones [7,8]. In both server and mobile applications, the increasing problem complexity (in terms of size and function) challenges the energy efficiency of DNN implementations.

Ultra low power DNN can be realized by dedicated hardware implementations. When DNNs are implemented as specialized hardware, its total energy consumption is often dominated by off-chip memory accesses to the information of synaptic connection since hardware specialization reduces the energy consumption of neuron computation [11,12,14]. Moreover, this trend is expected to be aggravated with the increasing problem complexity (i.e., the size of DNNs) because large DNNs for more complex problems require larger sets of synaptic connection information thereby increasing energy consumption of the off-chip memory.

In order to address this problem and facilitate more energy-efficient scaling of DNNs, our goal is to reduce the energy consumption of dedicated hardware DNNs. For this problem, we made a key observation that there are many cases where DNNs with smaller sizes (which we call little DNNs) can equally perform well compared to big DNNs. Based on this observation, we propose a big/LITTLE DNN (BL-DNN) which consists of little and big DNNs. Given an input for inference (e.g., an input image for object recognition), the little DNN is first executed to give an inference result. If the result is believed to be accurate, then it is taken as the inference result. If it is not certain to take it as the final result, then the big DNN is utilized to give the inference result.

The primary benefit of BL-DNN is energy efficiency. Between the two DNN implementations, the little DNN has the benefit of low energy consumption since, compared to the big DNN, its execution involves fewer synaptic information retrievals from the off-chip memory and a smaller amount of computation. Thus, as long as the execution of big DNN can be minimized, BL-DNN can give significant reductions in energy consumption compared to the conventional big-only DNNs.

However, determining when to use the big DNN is a challenging problem. This is because BL-DNN should be able to estimate whether the result from the little DNN is correct or not *without executing the big DNN*. Overly frequent false positives (i.e., considering the wrong inference from the little DNN as correct) will degrade inference accuracy. On the other hand, overly false negatives (i.e., considering the correct inference of little DNN as wrong) is also problematic since it will lose opportunities to reduce energy due to the redundant execution of energy-hungry big DNN. In order to tackle this problem, we propose utilizing the score information of the classifier layer, which has often been overlooked in conventional DNN approaches.

In summary, we made the following contributions in this paper:

978-1-4673-8321-9/15/$31.00 ©2015 IEEE

- We propose a novel architecture of big/LITTLE DNN that utilizes a conventional big DNN and its smaller version at the same time to optimize energy efficiency.
- We demonstrate that the score information of the classifier layer can be an excellent proxy for evaluating the accurateness of DNN execution results and propose utilizing the *score margin* as the evaluation metric.
- We present a design-time (static) method to determine the threshold of score margin which aims at minimizing the frequency of big DNN executions while reducing accuracy loss.
- We propose a runtime (dynamic) method to fine-tune the evaluation criterion which enables input data and DNN-specific tuning of BL-DNN execution.
- We evaluate our proposed BL-DNN with two real-world benchmarks, *MNIST* and *ImageNet*.

This paper is organized as follows. We review related work in Section 2. We explain our observations in Section 3. We describe the proposed BL-DNN in Section 4. We report our experiments in Section 5. We conclude the paper in Section 6.

2. RELATED WORK

Recently, hardware acceleration for DNN is gaining more and more attention. In [9], Chakradhar et al. propose a configurable hardware accelerator which can be adapted to different types of parallelism (convolution and intra/inter-output parallelism) in neural networks. In [10], Pham et al. propose a tile-based architecture for DNN execution. Each tile is equipped with full functional components for neuron and its configuration can be determined to meet the given DNN specification, e.g., the number of layers, layer size, layer type (e.g., convolution and pooling), etc. In [11], Chen et al. present a hardware implementation of neural networks called DianNao where a neuron is implemented with multipliers, a tree structure of adders, and local memory components for input data, synaptic weights, and output. They report that, when the hardware accelerator is utilized, the off-chip memory becomes the dominant source of energy consumption in DNN execution. In [12], the authors propose utilizing embedded DRAM in order to achieve performance improvement (via increased memory bandwidth) as well as energy reduction by avoiding costly off-chip accesses. In [13], Park et al. propose a low power hardware IP core called K-Brain which can perform both supervised/unsupervised learning and inference.

In [19], Peemen et al. propose a hardware architecture which tries to minimize off-chip memory accesses by adopting a similar approach to tiling. It aims at maximally reusing the data fetched from off-chip memory thereby improving both energy efficiency and performance. In [15], Lee et al. propose optimizing the bit width of neurons for energy reduction.

Some of previous work has explored a trade-off between inference accuracy and energy consumption. In [20] and [21], simplified adder models are proposed in order to achieve power savings with a negligible loss of output quality. In [29], Du et al. propose an approximate multi-layer perceptron with inexact multipliers which reduce energy consumption at a cost of degraded accuracy. In [30], Venkataramani et al. propose a systematic method to transform a neural network into an approximate neural network (AxNN) using the backpropagation technique by identifying each neuron's impact on output accuracy. In [14], Zhang et al. propose neuron criticality analysis to identify less influential neurons and a method to remove such neurons thereby giving energy reduction at a small loss of accuracy. Our proposed BL-DNN is different from these approaches since they do not exploit multiple DNNs for energy efficiency.

In the area of deep learning architectures, there are several approaches that utilize multiple neural networks in a hierarchical, coarse-to-fine, ensemble or cascade construction of neural networks [6,16,17,33]. In [6], Szegedy et al. propose a hierarchical neural network utilizing multiple instances of a sub-network called inception module. In face recognition, coarse-to-fine approaches adopt multiple neural networks [16,17]. For instance, the entire face boundary is determined by the first stage network while the networks at later stages identify each of sub-parts in the face, e.g., eye, mouth, nose, etc. In the ensemble approach, the integration of inference scores of multiple neural networks produces the final inference result [33]. For instance, in [6], seven DNNs are utilized as an ensemble to give the final inference result. In the above-mentioned works adopting multiple neural networks, all networks always participate in inference thereby incurring significant energy consumption.

Our proposed method belongs to the cascade (a.k.a. serial or sequential) approach [33]. In this approach, the classifier at an early stage (little DNN in ours) can produce the final inference result or reject the input thereby avoiding the execution of classifier(s) at the later stage(s). Most of existing cascade methods present training methods for the cascade, i.e., little DNN learning in our case [33,34,35,36]. In this paper, we show that a cascade of existing little and big DNNs can give significant reduction in energy consumption with a small loss of accuracy. In addition, we present (1) a design-time method which gives a static threshold (which determines the acceptance of the inference result of the little DNN) minimizing both the execution of energy-consuming big DNN and the loss of inference accuracy and (2) a runtime method (considered as a method of dynamic domain adaptation) which adjusts the threshold to adapt to the given input data sets. In both methods, we can make a trade-off between energy consumption and inference accuracy.

3. OBSERVATIONS

In this section, we explain our observations from real DNNs for *ImageNet* [22] and *MNIST* [25].

Key observation: Score margin can be a metric to judge successful inference

Typically, a DNN consists of multiple layers of feature extraction (e.g., convolution and pooling layers) and a (set of) classifier layer(s) [2,3]. Our key idea is to exploit the neuron output (which we call *score*) of the last classifier layer in estimating the accurateness of DNN execution. More specifically, we found that the successful inference of DNN execution can be known from the first-order score margin (in short, score margin), which is defined as follows:

$$\text{Score margin} = 1^{st} \text{ score} - 2^{nd} \text{ score}$$

where 1^{st} (2^{nd}) score represents the largest (the second largest) score value at the last classifier layer, respectively.

The rationale behind our usage of score margin is that as the 1^{st} score gives a larger margin against the 2^{nd} score, it is more likely that the top-1 inference is correct. As an empirical evidence of it, Figure 1 shows the relationship between the score margin and the inference success (a) and failure (b). We obtained the data by running Very Deep 19 [22] as a big DNN and VGG_F [23] as a little DNN with 50,000 images from the *ILSVRC2012* database.

Note that the computation cost (in terms of the number of multiplications in the convolutional layers) of the little DNN is significantly small and amounts only to 7.1% of that of the big DNN.

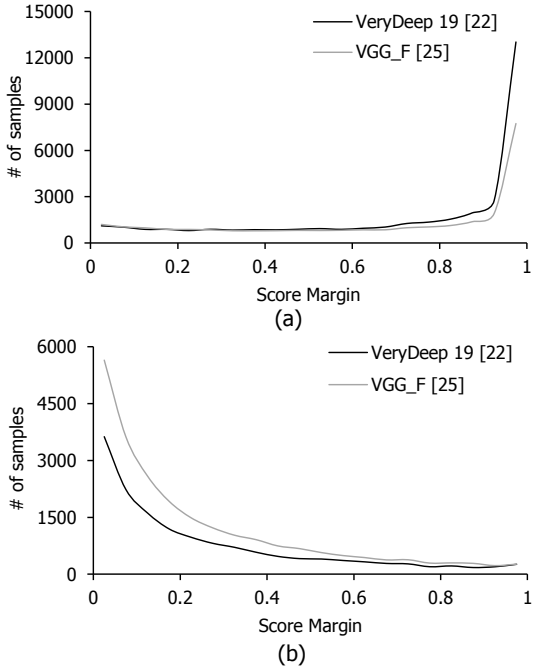

Figure 1. Histogram of (a) successful and (b) failed inferences among 50,000 validation samples at *ILSVRC2012* database with big and little DNNs.

As Figure 1(a) shows, the majority of successful inference gives large score values. This indicates that the score margin of the classifier layer can be an excellent proxy for evaluating the successful inference of DNN execution. Figure 1(b) also supports this argument. As shown in the figure, the majority of failed inferences yield small score margin. In other words, by checking only the score margin, we can estimate the success or failure of inference at a high probability.

We estimate the success/failure of inference using the score margin. If the score margin is larger (smaller) than a threshold, the inference result of DNN is considered as correct (incorrect). Applying the threshold to the score margin gives a binary classifier which we call score margin based result classifier (SMRC). Figure 2 shows receiver operating characteristic (ROC) curves[1] for the SMRCs of big and little DNNs. In the ROC curve, the larger the area under ROC gets, the better the binary classifier is. The ROC curves in Figure 2 are obtained by varying the threshold from 0 to 1. The area under the curve of SMRC is 0.826 for the little DNN (VGG_F). The area difference between the big DNN, VeryDeep 19 and the little DNN, VGG_F is only 0.031. In the case of *MNIST*, the area under the curve of SMRC is 0.945 for the little DNN (m8) and the difference is 0.034 between the area under the curve of the big DNN, LeNet and that of the little DNN (a little DNN used in our experiments as explained in Section 5). Figure 2 shows that SMRC is a fairly good classifier for determining the success/failure of classification. In addition, the figure also shows that the SMRC with the little DNN, VGG_F or m8 is a fairly good classifier. In terms of the real implementation, the SMRC has a benefit since the score margin is easy to obtain and, according to our analysis, it performs better than other possibilities such as the score itself.

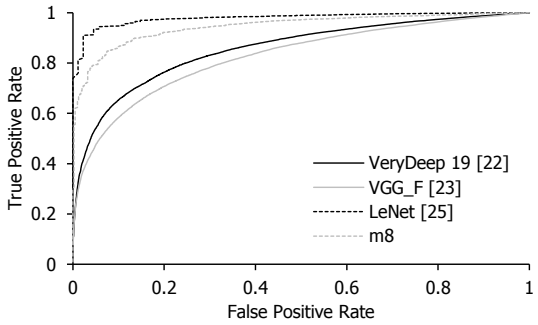

Figure 2. ROC curve of SMRC.

In summary, our observations demonstrate that the score information of the last classifier layer (i.e., score margin) can be utilized to evaluate the successful inference of little DNN execution. Based on that, we propose a novel DNN architecture which takes as many successful inferences as possible from the little DNN thereby minimizing the execution of big DNN.

4. BIG/LITTLE DNN

4.1 Architecture

We assume that each of little and big DNNs is trained by supervised learning. For the construction of each of little and big DNNs, any construction methods can be adopted, including unsupervised pre-training and supervised fine-tuning [1]. Given the two DNNs, our proposed BL-DNN is constructed as shown in Figure 3. It consists of a little DNN (top), a big DNN (bottom) and a success checker (a diamond in the figure). Our scheme does not assume particular big/little DNN designs, but is rather developed as a general framework for exploiting the trade-off relationship between two DNNs with different energy efficiency and accuracy characteristics (see Section 4.4 for the related discussion).

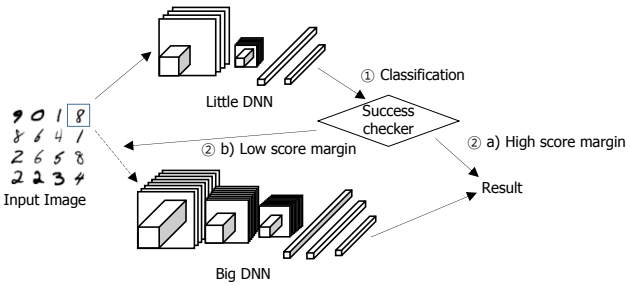

Figure 3. big/LITTLE DNN architecture.

The BL-DNN works as follows. Given an input, e.g., a handwritten letter as shown in the figure, the little DNN is first executed to give an inference result. Then, the success checker evaluates it and makes a decision on whether the big DNN needs to be executed for the final inference result or it is safe to take the inference result of the little DNN as the final one from the accuracy perspective. The success checker performs this decision by using an SMRC. Thus, if the score margin is larger than the threshold, the inference result

[1] ROC curves are conventionally used for representing the performance of binary classifiers.

from the little DNN is considered as correct and taken as the final inference result.

The key to the effectiveness of our architecture is on determining a good threshold for the success checker. An unnecessarily high threshold can increase energy consumption compared to the best threshold since both little and big DNNs are executed for inference in the most cases. On the contrary, an overly small threshold can hurt accuracy since most of the execution will rely only on the inference results from the little DNN, which are not accurate enough in some cases compared to the big DNN. Worse, the best threshold in terms of energy efficiency and accuracy depends on the input data as well as the DNN architecture (e.g., the number of layers, the number of neurons per layer, etc.). In the next subsections, we present a static method to determine the best threshold during design time (Section 4.2) and a dynamic method of adjusting the threshold to the given input data sets during runtime (4.3).

4.2 Static Threshold Method

The threshold of BL-DNN needs to be designed to achieve two goals as follows.

(1) To minimize the execution of big DNN

(2) To minimize the loss of inference accuracy

The first goal of minimum execution of big DNN can be described as the following gain function (for energy consumption) by using indicator function $I[x]$ (which gives 1 if x is true and 0 otherwise) and the score margin.

$$Gain(\theta) = \sum_{i=1}^{N} I[SM_{little}^{i} \geq \theta] \quad (1)$$

where SM_{little}^{i} for i-th input, θ and N represent the score margin of little DNN, threshold, and the number of inputs, respectively. The smaller the threshold gets, the larger gain can be obtained.

We use a loss function (for inference accuracy) to model the second goal of minimum loss of inference accuracy as follows.

$$Loss(\theta) = \sum_{i=1}^{N} \left(\begin{array}{c} I\left[SM_{little}^{i} \geq \theta\right] * \\ \left(P\left(correct_{big}^{i}\right) - P\left(correct_{little}^{i}\right)\right) \end{array} \right) \quad (2)$$

where $P\left(correct_{big}^{i}\right)$ and $P\left(correct_{little}^{i}\right)$ represent the probability density function (PDF) of successful inference for the big and little DNNs, respectively. Figure 1 (a) illustrates the PDFs for two DNNs, Very Deep 19 and VGG_F. Note that the second goal favors a higher threshold to reduce the loss of inference accuracy, which is contradictory to the first goal which favors smaller threshold.

In order to obtain a balanced design point towards both goals, we formulate the problem of static threshold as follows.

$$Loss\ Max\ (LM) = \sum_{i=1}^{N} \left(P\left(correct_{big}^{i}\right) - P\left(correct_{little}^{i}\right) \right) \quad (3)$$

$$Static\ Threshold = \arg\max_{\theta} \left(\frac{Gain(\theta)}{N} - \lambda * \frac{Loss(\theta)}{LM} \right) \quad (4)$$

The first and second terms in (4) correspond to the first and second goals, respectively. Both terms are normalized to the number of inputs N and LM respectably since $Gain(\theta)$ and $Loss(\theta)$ have different dimensions. The hyper-parameter λ enables us to make a trade-off between energy consumption and inference accuracy. In (4), the optimal θ is dependent on λ. Thus, if inference accuracy is more favored than energy consumption, λ needs to become larger. In this paper, we set a desired accuracy target, i.e., the limit of additional error rate (e.g., 1% or 0.3%) for each data set and sweep λ to determine the one which satisfies the accuracy target while using the little DNN as much as possible (Section 5.2).

4.3 Dynamic Threshold Adjustment

Typically, both big and little DNNs are trained for a general data set such as *ImageNet*. Thus, when a BL-DNN consisting of such big and little DNNs is applied to a specific domain, e.g., images on animals or flowers, which are often subsets of a training set, the static threshold introduced in Section 4.2 will lose opportunities for further reduction in energy consumption. Our goal is to enable further reduction in energy consumption in an input adaptive manner. This can be considered as a dynamic version of the domain adaptation [37].

We propose a dynamic method of adjusting the threshold to the given input data set. First, we define the probability of successful inference (PSI), which is calculated as follows.

$$\mathrm{PSI} = \sum_{i=1}^{N} \left(\begin{array}{l} \mathrm{P}\left(\mathrm{correct}_{\mathrm{little}}^{i} \middle| SM_{little} \geq \theta\right) \\ +\mathrm{P}\left(\mathrm{correct}_{\mathrm{big}}^{i} \middle| SM_{little} < \theta\right) \end{array} \right)$$

The PSI represents the probability that the inference is correct if (1) the results from the little DNN are used when its score margin is greater than the threshold and (2) those from the big DNN are used otherwise. Thus, the loss of inference accuracy can be minimized by maximizing the PSI.

The challenge here is that the two conditional probabilities cannot often be calculated during runtime since the calculation requires user intervention (on the accuracy of inference result). Instead, we propose a linear model that approximates each of the conditional probabilities to be linearly proportional to the associated score margin. For example, $\mathrm{P}\left(\mathrm{correct}_{\mathrm{big}}^{i} \middle| SM_{little} < \theta\right)$ is considered to be linearly proportional to the score margin of the result from the big DNN, SM_{big}. Based on this linear approximation model, the PSI is approximated as follows.

$$\mathrm{PSI}_{\mathrm{Approx}} = \sum_{i \in \mathrm{IN}} p_i \quad \text{where } p_i = \begin{cases} \alpha \cdot SM_{little}^{i} & SM_{little}^{i} \geq \theta \\ \beta \cdot SM_{big}^{i} & SM_{big}^{i} < \theta \end{cases}$$

where IN is the input data set, α and β are fitting parameters called score margin weighting factors.

We formulate the problem of dynamic threshold adjustment as the one of maximizing the $\mathrm{PSI}_{\mathrm{Approx}}$. To help understanding of our concept, Figure 4 illustrates $\mathrm{PSI}_{\mathrm{Approx}}$ for two BL-DNNs of *ImageNet* and *MNIST* used in our experiments. The figure shows the sum of SM_{little} ('Little'), the sum of SM_{big} ('Big'), the sum of SM_{little} and SM_{big} ('Total') and $\mathrm{PSI}_{\mathrm{Approx}}$ ('Weighted') for each threshold. We obtain the curves by varying the threshold (X-axis). It can be seen that, as the threshold becomes larger, Total gradually shifts from Little to Big since more inputs are handled with the big DNN. $\mathrm{PSI}_{\mathrm{Approx}}$ forms a convex curve as shown in the figure.

Our goal is to find a threshold which maximizes $\mathrm{PSI}_{\mathrm{Approx}}$ during runtime. For this purpose, we propose a simple but effective method as shown in Figure 5. Given an initial threshold and

incoming inputs, we periodically sample inputs with two thresholds, the current (θ) and candidate ones ($\theta + \Delta$), and calculate $\text{PSI}_{\text{Approx}}$ for both thresholds. Then, we determine the next threshold by comparing the two $\text{PSI}_{\text{Approx}}$ values. More specifically, if the threshold of $\theta + \Delta$ gives larger $\text{PSI}_{\text{Approx}}$ than θ, we increase the threshold by Δ; otherwise, the threshold is decreased by Δ (lines 2-4 in Figure 5). The amount of threshold adjustment, Δ is adjusted to control the speed of hill-climbing (lines 5-10). As will be shown in Section 5, this simple approach works well in maximizing $\text{PSI}_{\text{Approx}}$.

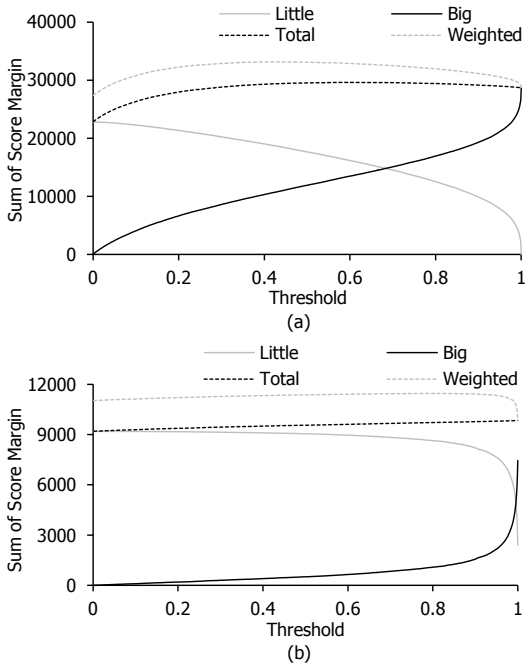

Figure 4. Sum of score margin vs. threshold: (a) *ImageNet* case and (b) *MNIST* case.

Initialize: $\boldsymbol{\theta = \theta_0, \Delta = \Delta_0}$

1. $\textbf{PSI}_{\boldsymbol{\theta}} \boldsymbol{= 0, \text{PSI}_{\theta+\Delta} = 0}$
2. **for(sample period)**
3. ⎪ **apply threshold to max** $\boldsymbol{(\theta, \theta + \Delta)}$
4. ⎩ **update** $\textbf{PSI}_{\boldsymbol{\theta}}$ **and** $\textbf{PSI}_{\boldsymbol{\theta+\Delta}}$
5. **if** $\textbf{PSI}_{\boldsymbol{\theta}} > \textbf{PSI}_{\boldsymbol{\theta+\Delta}}$
6. $\boldsymbol{\Delta = \Delta * -0.5}$;
7. **else if** $\textbf{PSI}_{\boldsymbol{\theta}} < \textbf{PSI}_{\boldsymbol{\theta+\Delta}}$
8. $\boldsymbol{\theta = \theta + \Delta,\ \Delta = \Delta * 1.2}$
9. **else**
10. $\boldsymbol{\theta = \theta + \Delta,\ \Delta = \Delta * -1}$
11. **Go to line 1 and repeat until finished.**

Figure 5. Dynamic threshold adjustment algorithm.

This simple approach for dynamic threshold adjustment performs well in the case where the sum of SM_{little} and SM_{big} ('Total') exhibits a convex curve as shown in Figure 4(a). However, we found that our linear approximation does not fit well in the case of *MNIST* (shown in Figure 4(b)) since its distribution of the sum of SM_{little} and SM_{big} ('Total') is concentrated heavily on the edge. In order to overcome this limitation, we can adjust score margin weighting factors α and β. For example, after setting α and β to 1.2 and 1, respectively, in Figure 4(b) ('Weighted' in the figure), the $\text{PSI}_{\text{Approx}}$ curve is turned into a convex shape from the monotonically increasing one, which lets our $\text{PSI}_{\text{Approx}}$ model achieve fast and stable convergence. The optimal threshold depends on the value of α. If α becomes larger, then the threshold gets lower (thereby giving larger reduction in energy consumption) and vice versa. Thus, the user can make a trade-off between energy consumption and inference accuracy by choosing appropriate score margin weighting factors.

4.4 Discussion

We propose a generalized framework that can leverage the trade-off relationship between big and little DNNs for energy efficiency and inference accuracy. Also, we will show in our experiments (Section 5) that the proposed approach can be robust in little/big configurations by showing that arbitrary combinations of existing little and big DNNs can give better energy efficiency than a solo big DNN execution.

We envision that there are several important issues to be addressed in the future work. Among them, one of the most important issues is the selection of big/little DNN configurations. Although we evaluate our architecture with different combinations of big/little DNNs to show its effectiveness as a general-purpose framework, we expect that systematic ways to construct the BL-DNN will ease its adoption. For instance, one can develop an automated way to transform a given big DNN into a little DNN in a way to maximize the energy efficiency under our framework. One possible option for this problem is to adopt the concept of transfer learning where a little DNN is trained with soft targets obtained from the big DNN [18]. We will elaborate further on this in our future work.

5. EXPERIMENTS

5.1 Experimental Setup

Tables I and II show the DNN examples used in our experiments. In the case of *ImageNet*, from the existing DNNs shown in Table I, we selected, as the big DNN, the largest one (called Very Deep 19 in [22]) and, as the little DNN, used each of the other smaller DNNs (VGG_S, VGG_M, VGG_F in [23] and CaffeNet in [24]). As shown in Table I, big and little DNNs are significantly different in terms of size while the difference of inference accuracy ranges from 8.6% to 16.3%. In the case of *MNIST*, we used, as the big DNN, an existing DNN available at Caffe [24], one of the implementations of LeNet [25]. We prepared little DNNs for *MNIST* by training smaller size DNNs as shown in Table II. Note that, in the case of *MNIST*, the difference of inference accuracy between little and big DNNs is small (between 0.2% and 3.1%). The maximum loss of inference accuracy is set to 1% at *ImageNet* and 0.3% at *MNIST*, respectively. By adjusting λ or α, both static and dynamic methods try to meet the specified target of maximum accuracy loss.

We implemented a hardware accelerator to process a neuron in Verilog as shown in Figure 6. We adopted a tile-based architecture in [10, 32] and ran maximum 512 neurons in parallel. As in [10, 11, 32], each hardware neuron is equipped with on-chip SRAM buffers for input, output, weights, and bias. For a given BL-DNN and SRAM size, we applied the optimization method in [32] to determine SRAM allocation to input/output/weight/bias buffers and obtained candidate configurations (SRAM allocations) having the fastest execution cycles. Among the candidates, the one having the minimum off-chip DRAM accesses is chosen. Due to the structure of BL-DNN, the single hardware accelerator needs to

Table II. DNN configuration for *MNIST*.

Name	VeryDeep 19 [22]	VGG_S [23]	VGG_M [23]	VGG_F [23]	CaffeNet [24]
Input	224x224 RGB	224x224 RGB	224x224 RGB	224x224 RGB	227x227 RGB
	Conv 3x3x64, pad 1	Conv 7x7x96, stride 2	Conv 7x7x96, stride 2	Conv 11x11x64, stride 4	Conv 11x11x96, stride 4
	Conv 3x3x64, pad 1	MaxPool 3x3	MaxPool 3x3, stride 2	MaxPool 3x3, stride 2	MaxPool 3x3, stride 2
	MaxPool 2x2	Conv 5x5x256	Conv 5x5x256 stride 2, pad 1	Conv 5x5x256, pad 2	Conv 5x5x256, pad 2, group 2
	Conv 3x3x128, pad 1	MaxPool 2x2	MaxPool 3x3, stride 2	MaxPool 3x3, stride 2	MaxPool 3x3, stride 2
	Conv 3x3x128, pad 1	Conv 3x3x512, pad 1	Conv 3x3 stride 1, pad 1	Conv 3x3x256, pad 1	Conv 3x3, pad 1
	MaxPool 2x2	Conv 3x3x512, pad 1	Conv 3x3 stride 1, pad 1	Conv 3x3x256, pad 1	Conv 3x3, pad 1, group 2
	Conv 3x3x256, pad 1	Conv 3x3x512, pad 1	Conv 3x3 stride 1, pad 1	Conv 3x3x256 1, pad 1	Conv 3x3, pad 1, group 2
	Conv 3x3x256, pad 1	MaxPool 3x3	MaxPool 3x3, stride 2	MaxPool 3x3,s stride 2	MaxPool 3x3, stride 2
	Conv 3x3x256, pad 1	InnerProduct 4096	InnerProduct 4096	InnerProduct 4096	InnerProduct 4096
	Conv 3x3x256, pad 1	InnerProduct 4096	InnerProduct 4096	InnerProduct 4096	InnerProduct 4096
	MaxPool 2x2	InnerProduct 1000	InnerProduct 1000	InnerProduct 1000	InnerProduct 1000
	Conv 3x3x512, pad 1				
	Conv 3x3x512, pad 1				
	Conv 3x3x512, pad 1				
	Conv 3x3x512, pad 1				
	MaxPool 2x2				
	Conv 3x3x512, pad 1				
	Conv 3x3x512, pad 1				
	Conv 3x3x512, pad 1				
	Conv 3x3x512, pad 1				
	MaxPool 2x2				
	InnerProduct 4096				
	InnerProduct 4096				
	InnerProduct 1000				
Top-1 Accuracy	69.41%	60.86%	55.95%	53.10%	58.90%

Table I. DNN configurations for *ImageNet*.

Name	LeNet [25]	Mini 1 (m1)	Mini 2 (m2)	Mini 3 (m3)	Mini 4 (m4)
Input	28x28	28x28	28x28	28x28	28x28
	Conv 5x5x20	Conv 5x5x10	Conv 5x5x7	Conv 5x5x2	Conv 2x2x2
	MaxPool 2x2	MaxPool 2x2	MaxPool 2x2	MaxPool 2x2	MaxPool 2x2
	Conv 5x5x50	Conv 5x5x25	Conv 5x5x15	Conv 5x5x5	Conv 2x2x5
	MaxPool 2x2	MaxPool 2x2	MaxPool 2x2	MaxPool 2x2	MaxPool 2x2
	InnerProduct 500	InnerProduct 250	InnerProduct 150	InnerProduct 50	InnerProduct 50
	InnerProduct 10	InnerProduct 10	InnerProduct 10	InnerProduct 10	InnerProduct 10
Top-1 Accuracy	99.12 %	98.93 %	98.85 %	97.96 %	97.47 %

Name	Mini 5 (m5)	Mini 6 (m6)	Mini 7 (m7)	Mini 8 (m8)
Input	28x28	28x28	28x28	28x28
	Conv 5x5x20	Conv 5x5x10	Conv 5x5x10, stride 2	Conv 5x5x10, stride 4
	MaxPool 2x2	MaxPool 2x2	MaxPool 2x2	MaxPool 2x2
	InnerProduct 500	InnerProduct 250	InnerProduct 250	InnerProduct 250
	InnerProduct 10	InnerProduct 10	InnerProduct 10	InnerProduct 10
Top-1 Accuracy	98.76 %	98.52 %	98.50 %	95.99 %

support two different DNNs. During the optimization process [32], we gave the same importance to the big and little DNNs. In terms of energy consumption, this optimization can give pessimistic results because the little DNN is always executed while the big DNN is rarely executed. The balanced optimization considering the execution frequency of big and little DNNs will be covered in future work.

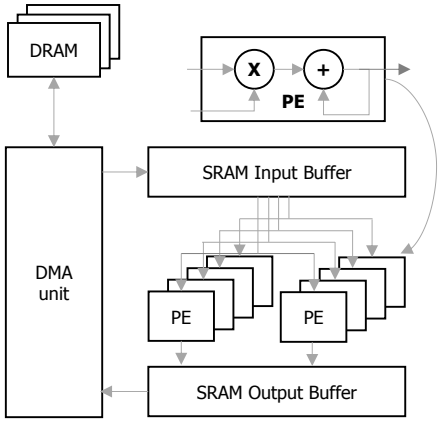

Figure 6. Hardware accelerator schematic.

We used the timing and power consumption data of the neuron obtained from Verilog synthesis (at 500MHz) with TSMC 65nm technology. We utilized CACTI to estimate the timing and power consumption of SRAM at 65nm technology. An off-chip DDR3 DRAM (4GB, 667MHz) is utilized to store synaptic weights. In order to model the power consumption of off-chip memory, we integrated our cycle-accurate simulator with DRAMSim2 [31]. The power and energy consumption is estimated for convolutional layers which consume most execution time and energy [32].

5.2 Experimental Results

Figure 7 shows the relationship between energy consumption and inference accuracy which is obtained by varying the static threshold from 0 to 1 and utilizing an SRAM of 16KB (*MNIST*) and 128KB (*ImageNet*), respectively. In the case of *MNIST* (*ImageNet*), we used Mini 8 (VGG_F) as the little DNN. As the figure shows, the smaller the threshold, the BL-DNN gives lower energy consumption since the frequency of big DNN executions is reduced. Energy reduction comes at a loss of inference accuracy. Thus, we need to make a trade-off between energy consumption and accuracy.

Figure 8 shows the inference results of BL-DNN in the case of *MNIST* dataset. 'Static' ('Dynamic) shows the result of static (dynamic) threshold method in Section 4.2 (4.3). The weighting factor (α) of the dynamic threshold method is selected to give the accuracy closest to the that of the static threshold method under the given limit on the accuracy loss (0.3%). We give the results (means and standard deviations) of 10 runs with shuffle the input data in random-order for each of static and dynamic methods since the result of static method depends on the training set selection to obtain the PDF (we selected, as the training set, 25,000 (5,000)

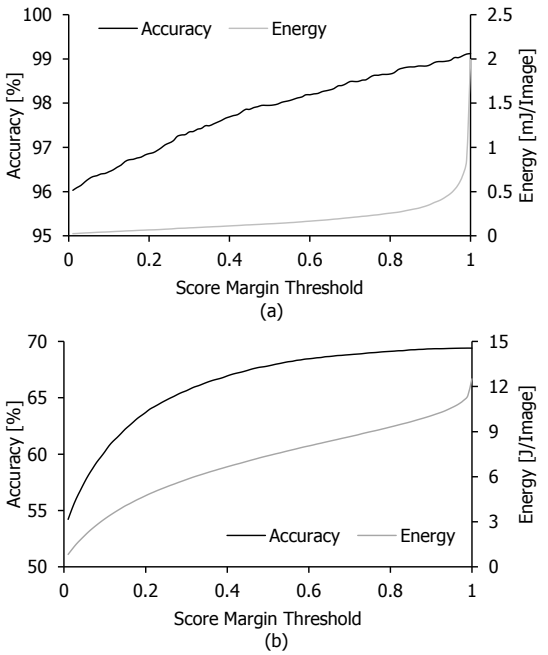

Figure 7. Energy consumption vs. accuracy for (a) MNIST (m8) and (b) ImageNet (VGG_F).

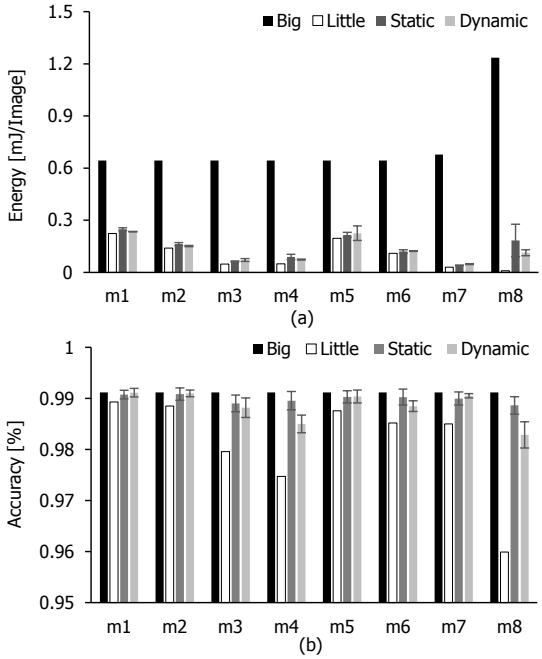

Figure 8. BL-DNN inference results of (a) energy consumption and (b) accuracy for *MNIST*

images from total 50,000 (10,000) validation sets in *ImageNet* (*MNIST*) case) and that of the dynamic method depends on the order of input data. 'Big' ('Little') represents a solo execution of big (little) DNN. As Figure 8 (a) shows, compared with the solo execution of big DNN, the static threshold method gives significant (average 80.0%) reduction in energy consumption under the average accuracy loss of 0.26%. The dynamic threshold method also gives average 81.0% reduction in energy consumption under the accuracy loss of 0.82%.

Note that our current methodology of BL-DNN design is to explore all the possible combinations of given big and little DNNs and select the best one which gives the maximum reduction in energy consumption under the given limit of accuracy loss. Thus, among the cases of Figure 8, the BL-DNN with Mini 7 will be chosen. It offers an energy reduction of 94.1% (93.1%) with an accuracy loss of 0.12% (0.07%) in the static (dynamic) threshold method. As our future work, we will work on design methodologies for the selection of good little DNN (offering large energy reduction while satisfying the given limit of accuracy loss).

Figure 9 shows the case of *ImageNet*. As shown in the figure, the static threshold method gives significant (average 40.6%) reduction in energy consumption with an accuracy loss of 0.94%. The dynamic threshold method gives larger reduction (average 42.5%) in energy consumption with an accuracy loss of 1.83%. In the case of CaffeNet, the static (dynamic) threshold method gives by 53.7% (47.5%) energy reduction with an accuracy loss of 0.90% (0.98%).

Comparing the results in Figures 8 and 9, BL-DNNs for *MNIST* and *ImageNet* exhibit different behavior in terms of energy consumption and accuracy. It is mainly because the difference of accuracy between little and big DNNs is smaller for *MNIST* than for *ImageNet*. Thus, in the case of *MNIST*, the little DNN tends to give large score margin thereby significantly reducing the frequency of big DNN executions to give larger reduction in energy consumption than in the case of *ImageNet*.

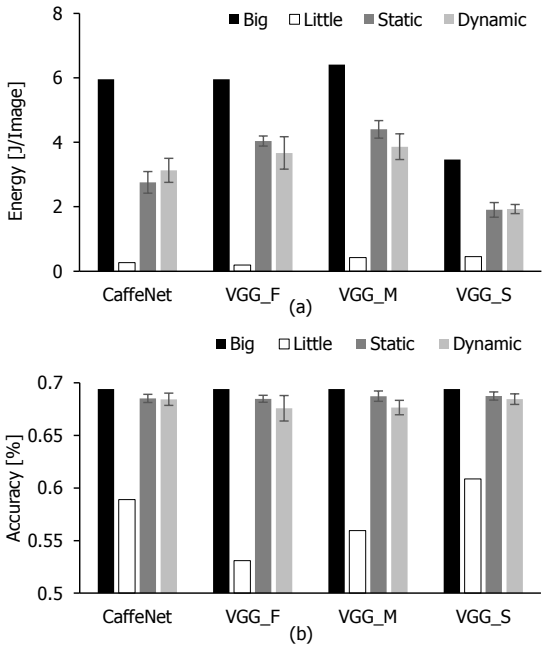

Figure 9. BL-DNN inference results for ImageNet.

Figure 10 shows the impact of SRAM on energy consumption. In both cases of *MNIST* (with Mini 7) and *ImageNet* (with CaffeNet), as SRAM size increases, energy consumption tends to decrease. It is mainly because off-chip DRAM accesses get reduced with larger SRAM. Given the same SRAM size (e.g., 16KB in *MNIST*), the BL-DNN gives significant reductions (e.g., 94.1%) in energy consumption. There are two exceptions in the trend of decreasing energy consumption, i.e., 32KB SRAM for *MNIST* and 16KB SRAM for *ImageNet*. It is because fast cycle configurations obtained by the optimization method [32], which try to unroll the loops in neuron computation for performance, often lead to large energy consumption due to increased DRAM accesses. In the case

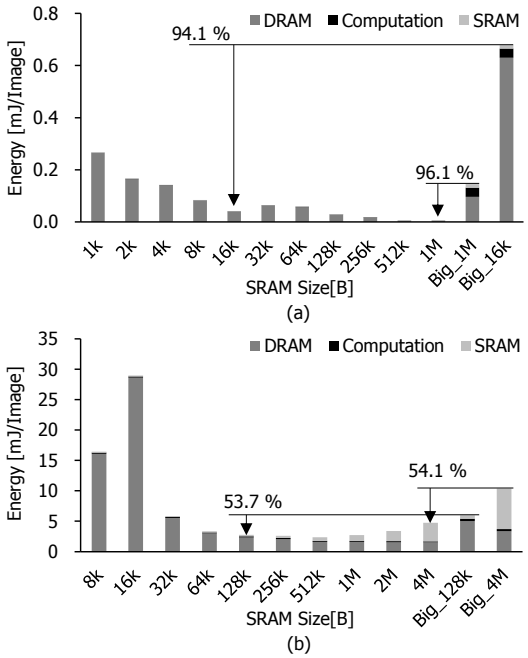

Figure 10. SRAM size vs. energy consumption of BL-DNN: (a) MNIST (Mini 7) and (b) ImageNet (CaffeNet).

of *ImageNet* in Figure 10 (b), as SRAM size continues to increase, the energy consumption of SRAM becomes dominant. It is because the increased energy consumption per SRAM access outweighs the diminishing benefit of reduced off-chip traffics.

6. CONCLUSION

In this paper, we propose a novel concept of big/LITTLE DNN (BL-DNN) which tries to minimize the execution of energy-hungry big DNN, thereby reducing energy consumption, while keeping inference accuracy. In order to judge the successful inference of little DNN execution, we propose utilizing the score margin at the classifier layer. We also presented static and dynamic methods of adjusting the threshold of score margin to the given BL-DNN and input data. Experiments with real DNNs show that our proposed BL-DNN offers up to 53.7% (*ImageNet*) and 94.1% (*MNIST*) reductions in energy consumption. As future work, we will work on the co-design of big and little DNNs in order to exploit the full potential of BL-DNN.

7. ACKNOWLEDGEMENT

This work was supported by the IT R&D program of MKE/KEIT (No. 10041608, Embedded System Software for New Memory-based Smart Devices), and Samsung Electronics.

8. REFERENCES

[1] G. Hinton et al., "Reducing the dimensionality of data with neural networks," *Science*, 313(5786):504-507, 2006.

[2] A. Krizhevsky et al., "Imagenet classification with deep convolutional neural networks," In *Proc. Advances in Neural Information Processing Systems (NIPS)*, 2012.

[3] A. S. Razavian et al., "CNN Features off-the-shelf an Astounding Baseline for Recognition," In *Proc. Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2014.

[4] P. Sermanet et al., "Overfeat: Integrated recognition, localization and detection using convolutional networks," arXiv:1312.6229, 2013.

[5] Y. Taigman et al., "Deepface: Closing the gap to human-level performance in face verification," In *Proc. Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[6] C. Szegedy et al., "Going deeper with convolutions," arXiv:1409.4842, 2014.

[7] J. Gehlhaar, "Neuromorphic processing: A new frontier in scaling computer architecture," Keynote Speech at International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2014.

[8] TeraDeep, Inc., http://www.teradeep.com/.

[9] S. Chakradhar et al., "A dynamically configurable coprocessor for convolutional neural networks," *In Proc. International Symposium on Computer Architecture (ISCA)*, 2010.

[10] P. H. Pham et al., "NeuFlow: Dataflow vision processing system-on-a-chip," In *Proc. Midwest Symposium on Circuits and Systems (MWSCAS)*, 2012.

[11] T. Chen et al., "Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," In *Proc. International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2014.

[12] Y. Chen et al., "DaDianNao: A machine-learning supercomputer," In *Proc. International Symposium on Microarchitecture (MICRO)*, 2014.

[13] S. Park et al., "93TOPS/W scalable deep learning/inference processor with tetra-parallel MIMD architecture for big-data applications," In *Proc. International Solid-State Circuits Conference (ISSCC)*, 2015.

[14] Q. Zhang et al., "ApproxANN: An approximate computing framework for artificial neural network," In *Proc. Design Automation and Test in Europe (DATE)*, 2015.

[15] Y. Lee et al., "Performance analysis of bit-width reduced floating-point arithmetic units in FPGAs: A case study of neural network-based face detector," *EURASIP Journal on Embedded Systems*, Jan. 2009.

[16] Y. Sun et al., "Deep convolutional network cascade for facial point detection," In *Proc. Conference on Computer Vision and Pattern Recognition (CVPR)*, 2013.

[17] E. Zhou et al., "Extensive facial landmark localization with coarse-to-fine convolutional network cascade," In *Proc. International Conference on Computer Vision Workshops (ICCVW)*, 2013.

[18] G. Hinton et al., "Distilling the knowledge in a neural network," In *Proc. NIPS Deep Learning and Representation Learning Workshop*, 2014.

[19] M. Peemen et al., "Memory-centric accelerator design for Convolutional Neural Networks," In *Proc. International Conference on Computer Design (ICCD)*, 2013.

[20] V. Gupta et al., "Low-power digital signal processing using approximate adders," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 32(1):124-137, 2013.

[21] R. Ye et al., "On reconfiguration-oriented approximate adder design and its application," In *Proc. International Conference on Computer-Aided Design (ICCAD)*, 2013.

[22] K. Simonyan et al., "Very deep convolutional networks for large-scale image recognition," arXiv:1409.1556, 2014.

[23] K. Chatfield et al., "Return of the devil in the details: Delving deep into convolutional nets," arXiv:1405.3531, 2014.

[24] http://caffe.berkeleyvision.org/.

[25] Y. LeCun et al., "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, 86(11):2278-2324, 1998.

[26] A. Krizhevsky et al., "Learning multiple layers of features from tiny images," Computer Science Department, University of Toronto, Tech. Rep, 2009.

[27] J. Deng et al., "ImageNet: A large-scale hierarchical image database," In *Proc. Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009.

[28] Micron, Inc., DDR3 SDRAM System-Power Calculator, http://www.micron.com/products/support/power-calc.

[29] Z. Du et al., "Leveraging the error resilience of machine-learning applications for designing highly energy efficient accelerators," In *Proc. Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2014.

[30] S. Venkataramani et al., "AxNN: Energy-efficient neuromorphic systems using approximate computing," In *Proc. International Symposium on Low Power Electronics and Design (ISLPED)*, 2014.

[31] P. Rosenfeld et al., "DRAMSim2: A cycle accurate memory system simulator," *IEEE Computer Architecture Letters*, 10(1):16-19, 2011.

[32] C. Zhang et al., "Optimizing FPGA-based accelerator design for deep convolutional neural networks," In *Proc. International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2015.

[33] M. Woźniak et al., "A survey of multiple classifier systems as hybrid systems," In *Proc. Information Fusion,* vol. 16, pp. 3-17, 2014.

[34] S. Venkataramani et al., "Scalable-effort classifiers for energy-efficient machine learning," In *Proc. Design Automation Conference(DATE)*, 2015.

[35] M. Termenon et al., "A two stage sequential ensemble applied to the classification of Alzheimer's disease based on MRI features," In *Proc. Neural Processing Letters,* vol. 35, pp. 1-12, 2012.

[36] A. F. R. Rahman et al., "Serial combination of multiple experts: a unified evaluation," In *Proc. Pattern Analysis & Applications,* vol. 2, pp. 292-311, 1999.

[37] H. Daume III et al., "Domain adaptation for statistical classifiers," In *Proc. Journal of Artificial Intelligence Research,* pp. 101-126, 2006.